NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

TASKMASTER: A PROTOTYPE GRAPHICAL
USER-INTERFACE
TO A SCHEDULE OPTIMIZATION MODEL

by

Stephen R. Banham

March 1990

Thesis Advisor Gordon H. Bradley

Unclassified

security classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report
2b Declassification Downgrading Schedule	Approved for public release; distribution is unlimited.
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*if applicable*) 37
7a Name of Monitoring Organization: Naval Postgraduate School	
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding Sponsoring Organization	8b Office Symbol (*if applicable*)
9 Procurement Instrument Identification Number	
8c Address (*city, state, and ZIP code*)	10 Source of Funding Numbers
Program Element No	Project No / Task No / Work Unit Accession No

11 Title (*Include security classification*) TASKMASTER: A PROTOTYPE GRAPHICAL USER INTERFACE TO A SCHEDULE OPTIMIZATION MODEL

12 Personal Author(s) Stephen R. Banham

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	March 1990	91

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Scheduling, Graphical User Interface, DSS

19 Abstract (*continue on reverse if necessary and identify by block number*)

This thesis investigates the use of current graphical interface techniques to build more effective computer-user interfaces to Operations Research (OR) schedule optimization models. The design is directed at the scheduling decision maker who possesses limited OR experience. The feasiblity and validity of building an interface for this kind of user is demonstrated in the development of a prototype graphical user interface called ***TaskMaster***. ***TaskMaster*** is designed as the Dialog component of a scheduling Decision Support System (DSS). The underlying scheduling model uses set-partitioning and mixed-integer linear programming to generate optimal schedules. Although the model was originally developed to address a specific problem, inter-deployment scheduling of Navy surface ships, ***TaskMaster*** has been designed to be problem-independent, enabling it to address a broad range of scheduling problems with the same general structure. ***TaskMaster*** demonstrates the type of interactive, graphical interface that can be developed specifically for non-specialists. It is easy to learn and to use, and yet fully exploits the power of a sophisticated OR scheduling model. The prototype is implemented on a NeXT computer, chosen for its advanced computational power and state-of-the-art graphical interface development tools.

20 Distribution Availability of Abstract	21 Abstract Security Classification
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified

22a Name of Responsible Individual	22b Telephone (*include Area code*)	22c Office Symbol
Gordon H. Bradley	(408) 646-2359	Code ORBZ

DD FORM 1473,84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

TaskMaster: A Prototype Graphical User Interface
to a Schedule Optimization Model

by

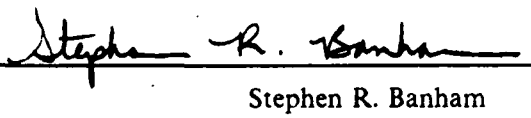
Stephen R. Banham
Lieutenant, Civil Engineer Corps, United States Navy
B.S., Oregon State University, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

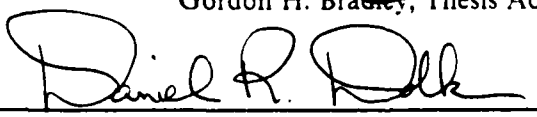
NAVAL POSTGRADUATE SCHOOL
March 1990

Author: 

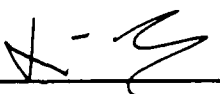
Stephen R. Banham

Approved by: 

Gordon H. Bradley, Thesis Advisor



Daniel R. Dolk, Second Reader



David R. Whipple, Chairman,
Department of Administrative Sciences

ABSTRACT

This thesis investigates using current graphical interface techniques to build more effective computer-user interfaces to Operations Research (OR) schedule optimization models. The focus of the design is the scheduling decision maker who possesses limited OR experience. The feasiblity and validity of this focus is demonstrated in the development of a prototype graphical user interface called ***TaskMaster***. ***TaskMaster*** is designed as the Dialog component of a scheduling Decision Support System (DSS). The underlying scheduling model uses set-partitioning and mixed-integer linear programming to generate optimal schedules. Although the model was originally developed to address a specific problem, inter-deployment scheduling of Navy surface ships, ***TaskMaster*** has been designed to be problem-independent, enabling it to address a broad range of scheduling problems with the same general structure. ***TaskMaster*** demonstrates the kind of interactive, graphical interface that can be developed specifically for non-specialists. It is easy to learn and to use, and yet fully exploits the power of a sophisticated OR scheduling model. The prototype is implemented on a NeXT computer due to its advanced computational power and state-of-the-art graphical interface development tools.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Within the computer industry, the 1980's may very well be characterized as the decade of the end-user. This period has seen the shift of computing power away from large centralized facilities controlled by computer professionals, to desktop computers operated by a wide variety of users, many with limited formal computer training. The operating systems and software that were developed during this period were increasingly oriented to non-computer specialists. The terms "user-friendly" and "user-interface" became common buzzwords within the computer industry. The use of computing power by an expanded group of users has increased the demand for powerful applications in a variety of disciplines. Tremendous advances in computer hardware have also contributed to the demand for software applications which take advantage of these hardware capabilities. As a result of these changes, new markets are developing for a variety of computer applications that were once limited to select audiences. A number of these software applications will undoubtably come from the field of Management Science Operations Research (MS OR). Although some software packages have been developed for MS OR, they have yet to gain widespread acceptance.

A. THE USER

Every tool for humans must address at least two aspects within its design. The first is concerned with the task that the tool is intended to perform. If the tool is a hammer, concerns of this type might include the size, weight, and hardness of the head as they relate to the ability to drive nails. The second aspect is concerned with the capabilities of the typical person using the tool. Again using the hammer for illustration, the focus would be on the handle, the way it fits the user's hand and the amount of mechanical advantage afforded by its length. Obviously if the typical user had no hands, hammers would be designed significantly different.

Computer software as well must address both these aspects if it is to be a useful and effective tool. The focus of this thesis is on that second aspect, the "user-tool" interface. Of great importance to the design of all software is the identification of the target audience or end-user. The application that is designed must take into account the abilities of those end-users. Therefore, one of the first questions that must be answered is, "who are the end-users of this software, and what are their capabilities?"

1. The Traditional User

Historically MS/OR computer applications were directed primarily at the OR specialist as the end-user. The user interfaces to these applications presupposed considerable understanding of the mathematical models upon which they were based. The interface was generally an afterthought, added on top of the model only after it was completed. As a result, the interface technology employed by most of these applications was rudimentary at best, requiring not only a thorough understanding of the model, but, often an intimate understanding of the computer program as well. Frequently these applications required special formatting of data for input and produced output that required reformatting before delivery to the manager. Firms wishing to employ these tools were required to work through an MS/OR department which developed and operated the applications. In general, only those who were MS/OR specialists possessed the requisite understanding of the mathematical and computer shorthand employed by these applications. This approach greatly limited the use of MS/OR solution techniques. Many firms could not afford to employ their own MS/OR specialists. Others were naturally reluctant to apply a tool that they did not begin to understand. This reluctance is further demonstrated in this quotation,

> "History shows that model based assistance is used all too infrequently by managers and policy-makers. Often this is the case because available modeling systems are incomprehensible to non-specialists in management science operations research (MS/OR). Managers may feel overly dependent on these MS/OR practitioners who more fully understand the underlying concepts of the modeling systems. Managers avoid this dependency by avoiding the very modeling systems that could enhance their decision-making capabilities." [Ref. 1: p. 3]

As Woolsey says, "people would rather live with a problem they cannot solve than accept a solution they cannot understand" [Ref. 2: p. 169].

Working through a specialist not only proved to be undesirable, but also introduced all the inefficiencies common to most intermediary relationships. The primary problems resulted at the interface which was created between the person who understood the specifics of the problem and the person who understood the models. This interface was exercised a minimum of two times and more commonly many more. Each translation from one domain to the other was generally cumbersome and fraught with opportunities for errors resulting from misinterpretation.

This traditional approach, especially in the light of the current trends in computing, is as unnecessary as it is undesirable. There is currently the potential for converting a number of MS/OR applications into software packages which are capable of

being used directly by the people who depend on the output information. This will, however, require an entirely different approach to the development of MS OR application software. The need for this new approach is further demonstrated through the use of an analogy which follows.

2. An Analogy

A mathematical model can in many ways be compared to an automobile engine. The mechanic is interested in the technical capability of the engine in much the same way that the MS OR specialist is interested in the features of a given model. Most users however, are less concerned with the specifics of the engine and more concerned with the way it functions as part of a complete car. Very few operators of cars understand all the intricacies of the engine's operation. Rather, the capabilities of the engine have been extended to user-oriented objects within the automobile that can be used to exploit its power. Those things which are attached to the engine enabling it to perform a specific function are analogous with the user interface. Many mathematical models today exist as little more than engines which evoke the admiration of other mechanics who understand and appreciate their elegance. Most contain such basic interfaces that they can only be understood and operated by those with special training and hence often the actual user must work through a separate MS OR department to use a model. This is analogous to designing cars that require mechanics as chauffeurs.

While it is probable that there will always be research and development into new and better models just as there is into engines, it is time that some of these models be developed into "stock" rather than just "custom" applications. Not only will this provide access to a wider audience, it will also provide needed feedback to the design process.

There are some objections to the adaptation of models for general use. Often these objections are based upon valid concerns about abuse or misuse of models stemming from a lack of understanding on the part of the user. Similar concerns were raised during the early days of the automobile. The development of any powerful tool introduces a risk associated with its possible misuse. If the tool, however, has application to a broad base of users, the benefits usually outweigh the risks. Further, these risks should encourage the designers to incorporate into their designs aids and mechanisms which encourage safe and informed user operation. One of the keys to minimizing this risk is to build user-oriented representations into the interface that facilitate proper use and educate the user in the solution methodology.

A movement is beginning which encourages the development of MS OR models into end-user applications. This movement is being spearheaded by managers who have

experienced firsthand the power of mathematical models and OR practitioners who recognize this as the future of OR. From their inception the "mechanics" of OR have concerned themselves almost exclusively with the design and construction of newer and more powerful "engines". In the past decade there has been an increasing awareness that these "engines" must be incorporated into fully functioning automobiles if they are ever going to go anywhere.

3. The Proposed User

Once it has been accepted that certain MS/OR models can be effectively used directly by other than MS/OR specialists, it remains to define more specifically the capabilities of this audience. There are two principle assumptions which must be made regarding the managers and decision makers proposed here as the end-users.

The first assumption concerns computer literacy. Recognizing that this opportunity for direct usage has been facilitated in part by the proliferation of personal or desktop computers, it is natural to presuppose that the user is able to use these computers as a platform for these applications. The user in this instance is assumed to have a good understanding of the basic operations associated with a computer and in particular be able to use a graphical user interface. The graphical interface itself is an outgrowth of an end-user focus at the level of the operating system. Later in the paper the reasons for selection of this type of interface will be discussed more fully. This type of interface is clearly the wave of the future and is being implemented on all major hardware platforms. The most important benefit in this discussion is the ease with which the necessary degree of computer literacy is attained. Willingness on the part of the decision maker to use a computer is probably the largest hurdle. With each succeeding generation receiving greater exposure to computers and at earlier ages, this hurdle is clearly diminishing.

The second assumption is that the user is familiar with the details of the specific problem being addressed. If the persons operating the system don't understand the nature of the problem they are not in the target audience, for the primary reason for proposing an application of this sort is to bring together the specific knowledge of the problem domain and the tool used in its solution. It is therefore fundamental to the design of the application that this problem domain knowledge, when coupled with basic computer literacy, be sufficient for use of the application. The system itself must be capable of augmenting this domain knowledge with any additional insight required in the solution process. It is also important to note at this point that these applications will necessarily rely more heavily on an understanding of the basic structure of the problem

than on the particulars. By focusing on the underlying structure an application is able to address a much broader base of problem domains. This generic aspect is addressed further in the "Scheduling" paragraph below.

B. A DSS APPROACH

The concept of putting models directly in the hands of the decision maker is not original. A whole discipline within the Information Sciences committed to developing systems for the decision maker has begun to flourish in the past decade. These software systems are referred to as Decision Support Systems (DSS). The following is a widely accepted definition of a DSS:

> "Decision Support Systems (DSS) are interactive computer based aids designed to assist managers in complex tasks requiring human judgment. The aim of such systems is to support and improve a decision process." [Ref. 3: p. 21]

The increasing popularity of DSS is certainly related to the decentralization of computing power and the related growth in end-user computing. The power of a DSS is in its ability to bring together data and sophisticated models in a single package that can be used directly by decision makers to address their problems.

The OR MS community recognizing the direction of computing in recent years has taken steps to align itself more closely with the user-oriented field of Decision Support Systems (DSS). This strategy is documented in an article entitled "Can MS OR Evolve Fast Enough?", based upon the plenary address given by Geoffrion to the EURO V - TIMS XXV International Meeting in 1982. Geoffrion identified the following two advantages which characterize the DSS style [Ref. 4: p. 21]:

- "It places a high value on flexibility of system use and adaptability to changing user needs."
- "It puts the user first and the underlying technology second, with particularly careful attention to the user interface."

A similar theme is echoed by Jones who observes that, "One of the principle contributions of the work of DSS has been its emphasis not on algorithms (unlike OR), but on the infrastructure, e.g., the representations, surrounding the problem solving process" [Ref. 5: p. 892].

From another perspective a DSS is nothing more than the next logical extension in the development of software systems. Software has always in some way supported decision processes. The DSS simply integrates into the software more of those functions which historically were performed externally by the human user.

The DSS can be thought of as containing three interrelated technical capabilities which define three major components or subsystems [Ref. 6].

- The Dialog Subsystem
- The Data Subsystem
- The Model Subsystem

The model component includes decision models that can be used to analyze problems of interest to the decision maker. This is the portion of the application which contains the particular mathematical model(s) used in the decision process. In the previous analogy this would be the "engine". Obviously, a model should be selected which is well suited to the type of problem the decision maker is facing. Again resorting to the analogy, it is nice to have a "powerful engine under the hood".

The data component manages the raw material or "fuel" used in the decision making process. It includes a data base for storing the data and a management system which oversees creation, maintenance, access, update, and protection of the data. In much of the DSS literature the data component includes more data than is specifically required by the model, often providing access to the entire corporate database. This thesis will take a more restricted view of the data component focusing only on that data required to support the underlying MS/OR model. The way the data is stored is also referred to as a model. The particular model this paper will assume is the "relational model". This particular model is widely accepted as one of the most functional, addressing the broadest range of naturally occurring relationships. Within the relational model data is viewed as a series of two dimensional tables of related information. Each row is referred to as a "tuple" and each column as a "attribute". Each individual value is a "field". Beyond this basic understanding very little time will be spent on the discussion of this component.

This thesis will emphasize the third component, the dialog subsystem. The dialog component is the most critical in supporting the direct use of the system by the decision maker. This component may be viewed as the part of the system that the user works with to exploit the capabilities of the other two components. For this reason it is understandable that, "from the DSS users' point of view, *the Dialog is the System*" [Ref. 6: p. 29]. The importance of the dialog subsystem in model-based DSSs is shown in this quote from Brennan and Elam,

> "The DSS movement has highlighted the fact that effective decision making aided by models requires a software environment that includes more than a sophisticated

solver. It requires a user interface that allows managers to define models and to view their results in a framework-a conceptual model-that makes sense to them. The widespread acceptance of micro-based speadsheet packages (e.g., Lotus 1-2-3) is in part due to the fact that these software packages allow the user to work in a modeling environment that is a familiar one. The lack of user-oriented interfaces as opposed to technically-oriented interfaces has resulted in the less than enthusiastic response of decision makers who feel, justifiably, that they have no means of controlling or understanding the models being used." [Ref. 7: p. 131]

The importance of the user interface has been clearly evidenced within the marketplace. Some of the most significant changes and trends in commercial software and operating systems deal with the user interface. During the past several years there has been a significant migration to Graphical User Interfaces (GUI) and it is anticipated that, before too long, most operating systems will employ interfaces of this type. Perhaps in part fueled by these advances, users have come to expect and demand interfaces which are more intuitive and easier to use. Even the most powerful software package is poorly received if the interface to it is poorly constructed. On the other hand those packages which provide an easy to use interface along with powerful capabilities become industry standards. Clearly one of the earliest examples of this kind of success was the Lotus 1-2-3 spreadsheet. Later in this paper, the major components of the dialog experience will be identified and some principles for design will be developed.

It is often difficult to neatly separate a DSS into these three parts. Much of the impact of the DSS approach has been based upon the notion of balanced integration of dialog, data, and modeling technologies into one complete and powerful system. Therefore, even though the focus will be on the dialog component, it will be necessary to consider certain aspects of the other two. Traditionally, the design of these systems focused primarily on the other two components. The objective here will be to seamlessly integrate a more advanced technology in interface design with the previously developed capabilities.

C. THE SCHEDULING DOMAIN

Many DSSs have within them solution models from the fields of Management Science and Operations Research. Some of the first problems these disciplines sought to address were scheduling problems.

In an environment which increasing competes for scarce resources, the efficient allocation and scheduling of resources is becoming essential for survival. Not only is this true in the private sector, but, in government as well. In government and in larger commercial and industrial organizations, the complexity and scope of many scheduling

problems makes their solution intractable by the unaided human mind. These large and complicated scheduling problems are ideal candidates for the computational power offered by present day computers.

Over the past several decades, Operations Research has successfully applied a number of sophisticated and powerful models to the solution of certain classes of scheduling problems, yet few have gained widespread acceptance. For the reasons previously discussed, managers are often reluctant to embrace these model-based solutions, choosing instead manual solution methods which are time consuming and inexact.

By building scheduling applications directly for the end-user which incorporate sophisticated, yet understandable interfaces, it is anticipated that much of this reluctance will be overcome. The success of this approach has already been demonstrated by software packages using Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) models. Software packages have been developed for these techniques of scheduling which employ user-oriented interfaces enabling direct use by other than OR specialists. It is suggested that the same technique be used to extend other scheduling models directly to the user.

This approach is both specific and generic simultaneously. It is specific in that it contains a particular underlying algorithm based upon a specific mathematical model. On the other hand, it is proposed that the interface be designed in such a manner so as to permit a whole class of problems. In this respect they can be considered to be generic. This generic capability exist in the majority of CPM/PERT applications. The interface of these software programs have been designed to accommodate the specifics of any problem to which these decision factors can be applied. CPM/PERT software can be applied with equal effectiveness to the scheduling of ship construction and building construction. This can be characterized as the "engineering" as opposed to the "artistic" approach. The benefit of this approach is that it develops structures that are extensible to a variety of specific situations. It recognizes that within the scheduling domain the factors influencing scheduling decisions are often not unique to a single problem. The underlying algorithm or model remains unchanged. Only the user interface is affected. This enables these applications to be used within a wide variety of problem domains. This obviously extends the breadth of their appeal and has no doubt contributed to their success. This approach to building scheduling systems has recently been supported in a paper presented to the 1989 meeting of ORSA/TIMS. In that paper the authors make the following observation.

> With traditionally developed systems, the potential market was users with *exactly* the same problem. But, with problem-independent scheduling systems, the end user market is increased to include those with a *similar* problem. [Ref. 8: p. 2]

This is also consistent with the DSS approach which is often characterized by its emphasis on adaptability. A DSS generally "...places a high value on the flexibility of system use and adaptability to changing user needs" [Ref. 9: p. 21]. To the degree possible, this same emphasis on adaptability should be used in the development of other scheduling applications.

There are a couple of powerful scheduling models that have been developed for the Navy in recent years which have yet to be implemented. This thesis will focus on one that was designed by Wing for the inter-deployment scheduling of surface combatants called "SURFSKED". [Ref. 10] This application focuses on one particular problem, however, the underlying algorithm and math model contain some useful constructs which could be applied to a significantly broad class of problems. The author himself acknowledges that the "...method and model can be used by submarine, air, and marine units" [Ref. 10: p. 61] in addition to scheduling ships. The underlying algorithm and model will be retained, while adapting the interface into one which is simple and generic. This will result in the creation of a problem-independent scheduling model which can be used directly by end-users to solve a variety of similar scheduling problems from diverse domains. The present interface building tools offer the ability to achieve this goal with unprecedented ease and with impressive results.

D. THESIS APPROACH

This thesis looks at scheduling solution models developed by OR and the prospect of adapting these solutions into Decision Support Systems (DSS) which can be used directly by managers and decision makers who are not OR specialists. The most significant effect of this type of adaptation is upon that portion of the DSS dealing with the user interface and so this paper will necessarily focus on that aspect.

One objective of this thesis is to encourage decision makers, who have been hitherto reluctant, to embrace these scheduling models. The focus will be on this new group of end-users. This is a relatively untapped, direct market for OR modeling tools. It is suggested that this market can be reached by emphasizing the user interface in the development of scheduling DSS. Further, it is anticipated that by harnessing the power of a sophisticated mathematical model for use by the people familiar with scheduling, better scheduling solutions will result.

A second, and related objective, is to encourage those within the OR discipline who design these models to place more emphasis on their development for end-users who are not OR specialists. Even Wing, the author of SURFSKED recognized that the model needed to be modified to make it into an "end-user product" [Ref. 10: p. 61]. This thesis will demonstrates the feasibility of building an interface which is applicable to end-users in a variety of problem domains.

The remainder of the thesis is organized as follows.

Chapter II explores the nature of the scheduling problem and establishes a conceptual framework for scheduling. The chapter will then examine Wing's SURFSKED model, and from that develop a generic problem class that can be solved using the same basic model. This generic model will later form the basis of a prototype interface.

Chapter III will look at interface design considerations and their application to the construction of the dialog component of a scheduling DSS. The various aspects of the dialog experience will be explored for principles to guide the design of the prototype interface. Special emphasis is placed on good graphical technique, direct manipulation and other current human-computer interaction concepts.

Chapter IV will discuss the use of prototyping as a methodology for development of software systems and present ***TaskMaster***, a prototype interface for the input of data and constraints to a generic scheduling DSS.

Finally, Chapter V will present some of the observations that were made during the design and development of ***TaskMaster*** and make recommendations for future related study.

II. SCHEDULING

A. GENERAL

No one knows for certain when man first began to schedule, but clearly scheduling has its roots very early in history. Scheduling is an inseparable part of the daily activity of human planning. The need for scheduling is predicated upon the finiteness or scarcity of the items to be scheduled. The person engaged in scheduling is concerned with efficient use of those resources which are perceived to be the most scarce. Time itself is often limited and therefore viewed as a scarce resource which must be efficiently used. A good schedule assigns the limited resources according to goals and objectives prescribed by the scheduler. The measure of "goodness" is the degree to which those objectives are achieved and the schedule with the highest measure is considered to be optimal.

The goal of this chapter is to develop and present a generic scheduling model. However, in order to achieve that objective, it is first necessary to provide the reader with a basic understanding of the scheduling problem domain. This first step will focus on the development of a conceptual framework which encompasses the broadest scope of scheduling problems. Here the basic components of the scheduling problem will be defined and the various facets of the scheduling decision itself will be identified. Once that has been done, a particular scheduling problem and solution methodology will be explored in the light of the developed framework. Finally, the specific problem will be adapted into a generic and broadly applicable scheduling model which retains the basic solution methodology and underlying mathematical model of the specific problem.

B. A CONCEPTUAL FRAMEWORK

1. The Scheduling Environment

In order to understand the basic nature of the scheduling problem it is important to adopt a working definition of scheduling. Scheduling may be defined very simply as, "...the allocation of resources over time to perform a collection of tasks" [Ref. 11: p. 2]. This definition was selected because it identifies three different primary objects or entities that are common to a majority of scheduling problems; *resources*, *tasks*, and *time*.

Certain characteristics of resources, tasks, and time vary with the problem domain. Later in the development of a generic problem each of the three dimensions will

be defined more rigorously. The following abbreviated definitions are provided to establish a common basis of understanding because other terms are often used within specific problem domains.

Time provides the backdrop for most schedules. Unlike the other two dimensions it is by nature continuous rather than discrete. Most scheduling problems establish a period of time or window of time which limits the scope of the scheduling problem.

Historically, because much scheduling work was performed in manufacturing, resources were "machines" and much of the scheduling literature continues to uses this term to describe resources. In the conceptual scheduling framework, resources are the basic elements that are used by, required by, or satisfied by, tasks. Resources normally have associated with them characteristics which restrict their assignment to tasks and times.

The manufacturing environment has historically referred to tasks as "jobs" or "operations". Other names commonly given to tasks are "events" and "activities". Tasks are activities which require, use, or service resources for a particular time period. Tasks, like resources commonly have characteristics which dictate the resources and times that may be associated with them. The definition of task is the most critical to the conceptual framework developed within this chapter. This is because there are at least two other definitions which are predominant within scheduling literature. One definition associates a specific time period with each task. The other defines tasks to include the specific resources which they require. Both of these definitions are rooted in simplifications of the scheduling problem which will be discussed further in the following section dealing with the scheduling decision. In the framework developed here, the resources and timeframe that may be associated with a particular task are left as part of the scheduling decision rather than being included within the definition of the task. By providing a more basic definition of tasks this conceptual framework can accommodate both the simplifications as well as more complicated problems where these simplifications aren't applicable.

The terms *object* and *entity* were purposely used above to describe resources, tasks and time because of their use in other disciplines to describe the structure of models, databases, and interfaces. The use of these terms here provides insight into the role these three scheduling components have in the model, the interface, and the database. The objective is to provide a synergy of some of the latest techniques in each of the three DSS supporting disciplines.

The terms *object* and *object-oriented* have gained increasing popularity within both computer and information sciences. The term has been applied both in interface design and in relational database design. In the latter, it is used to describe "...a stored data representation of an entity of concern to the user" [Ref. 12: p. xix]. By identifying these three basic objects we establish not only a framework for the model but also for the organization of the underlying data as well. Later, in the discussion of interface design, graphical user interfaces will be discussed which make use of graphical objects and object-oriented programming.

Within the field of modeling, an attempt has been made by Geoffrion to identify a structure within solution models. His work of "Structured Modeling" breaks the model into a hierarchy composed of various types of "entities". At the root of the hierarchy are the "primitive entities" which he defines as, "...elements (which) have no associated value and generally represent things or concepts postulated as primitives of the model" [Ref. 9: p. 553]. Resources, tasks, and time, can be thought of as the primitive entities of a generalized scheduling model. By identifying these three as primitive entities, a framework is established which can be used to compare, and ideally combine, different scheduling models.

These three can also be thought of conceptually as dimensions. In this way the scheduling problem can be viewed as a three-dimensional space defined by axes of tasks, resources, and time. This dimensional connotation can provide insight into the scope of the specific scheduling problem being contemplated. The size of the particular problem domain is represented by the space with dimensions corresponding to the numbers of resources and tasks, and the time period under consideration. Generally the larger the space, the more complex the problem. This connotation also provides a method for visualizing schedules. A schedule represents selected portions within the domain space that identify the resources and tasks assigned together in time. A graphical depiction of this scheduling framework can be seen in the three dimensional Gantt Chart developed by Jones [Ref. 5: pp. 891-903]. A slightly modified version of this graphic is shown in Figure 1.

2. The Scheduling Decision

Having defined the scheduling framework as consisting of three objects, the scheduling decision is one that addresses the combination of these three into time scheduled and "resourced" events.

In relational database terminology, schedules represent **association objects**. An association object has been defined by Dolan & Kroenke as, "an object that documents

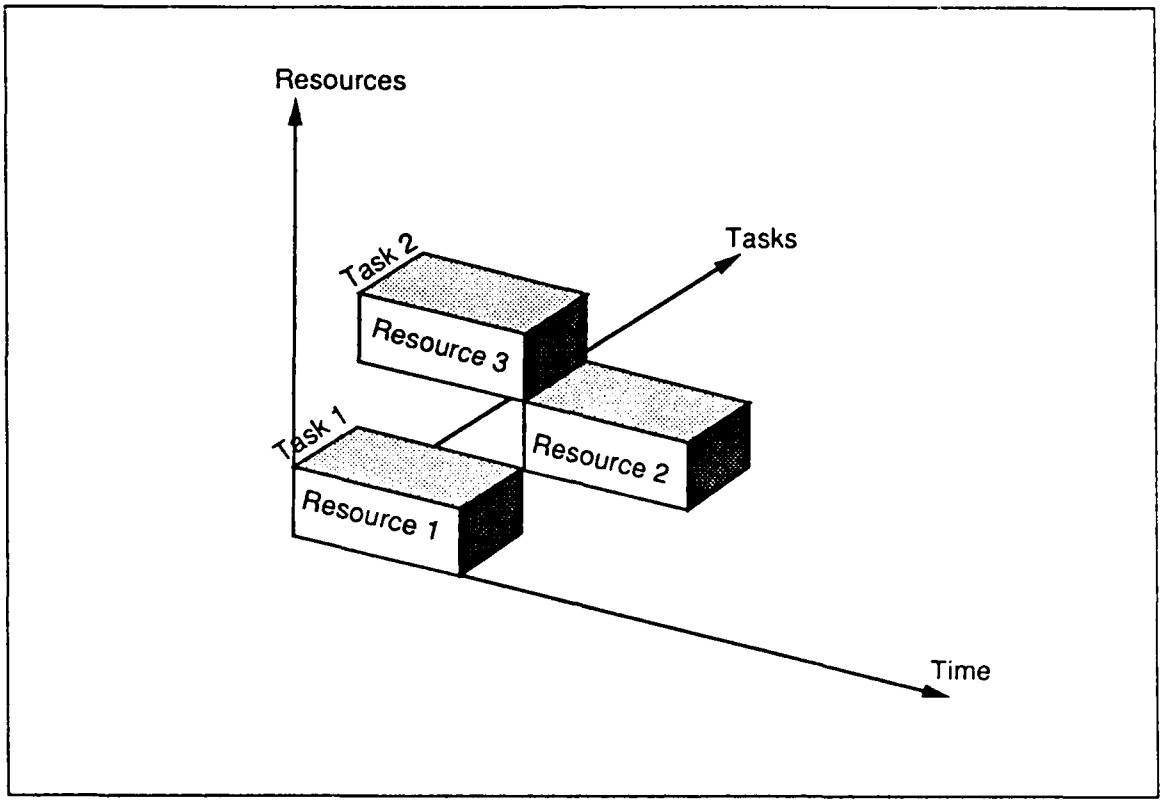


Figure 1. Three Dimensional Gannt Chart
Source: Adapted from [Ref. 5]

a relationship between two or more other objects" [Ref. 12: p. 195]. Schedules in this sense document the relationship between the three objects of tasks, resources, and time. A relational diagram depicting this relationship is shown in Figure 2. A completed schedule then represents a set of linking relationships. For example, a schedule might assign student A and student B (resources) to Chemistry 101 (task) during the fall quarter (time).

It should be noted that not all problems appear to be concerned with all three of these dimensions. This was previously alluded to in the definition of tasks. The composition of the primitives of a given scheduling model can be useful in providing insight into the complexity of both the decision and solution methodology. Certainly the most complex class of problems involves decisions about linking all three of these objects. However, many current solution models do not address that degree of complexity, but rather choose to combine two of the objects from the outset in order to

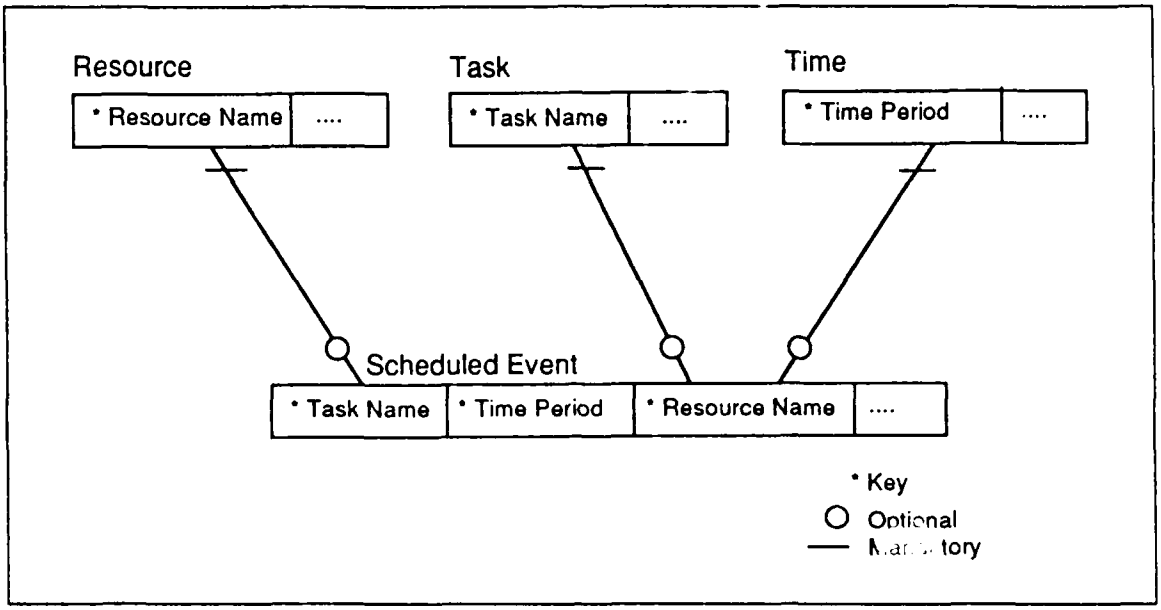


Figure 2. Relational Diagram for Scheduling Framework

simplify the problem. A common simplification is to combine tasks and time. Each task is identified with a particular time period and the problem becomes simply a matter of allocating (or linking) resources to these time-bound tasks. The second common simplification involves tasks which are by definition rigidly linked to specific resources. Good examples of this type of simplification are demonstrated in the popular PERT and CPM scheduling solution techniques which focus primarily on assigning tasks with their embedded resources to particular time periods. Both of these approaches, by combining two of the basic objects, have the effect of removing one degree of freedom from the overall scheduling decision. By keeping all three objects initially separate this framework is able to address the larger problem domain that includes these two smaller subsets.

Where there are few resources and tasks and the time period under consideration is small, or one of the previously discussed simplifications exist, the scheduling problem can be simple and intuitive, lending itself to manual solution methods. A large number of scheduling activities performed today are performed manually using a trail and error application of heuristics to arrive at an acceptable schedule. As the size and scope of the scheduling problem increases this particular method becomes increasingly more time and manpower intensive and offers no assurance that the selected schedule represents the best solution. In recent years the larger and more complex problems have been solved predominantly by the application of mathematical optimization models run

on computers. These apply "...a quantitative approach that begins with the translation of decision-making goals into an explicit objective function and decision-making restrictions into explicit constraints" [Ref. 11: p. 5]. From this statement we identify two basic components influencing the decision process, goals and constraints. These two will now be discussed in more depth.

a. Constraints

Constraints may be thought of as the rules which govern the specific scheduling problem. They are inviolable, that is, schedules that don't conform are not even considered. Constraints remove from consideration certain combinations of the three primary objects. They may do this explicitly by identifying those combinations which are inappropriate, or this may be done implicitly by identifying specific characteristics which must be met by all acceptable combinations. In this way, constraints reduce the range of possible schedule solutions to a smaller set of "feasible" schedules.

Baker has identified two popular categories of constraints, *allocation* and *sequencing* constraints. Here again the effect of the two predominant simplifications discussed earlier can be seen. Where tasks are defined to include specific times the decision process focuses on the question, "...which resources will be allocated to perform each task?" The constraints which restrict these assignments are *allocation constraints*. A typical allocation constraint will specify the types and numbers of resources required by a specific task, or viewed from the other perspective, the types of tasks that are required by the resources. Referring to Figure 3, these constraints identify those links which are feasible between each task/time combination and resource.

In the second type of simplification, where tasks are defined to include specific resources, the only decisions required are those which ask, "...when will each task be performed?" These time assignments are commonly restricted by *sequencing constraints*. Referring to Figure 4, these constraints identify those links which are feasible between each task/resources combination and time. The most common sequencing constraint involves prerequisite relationships, where the start of a task is contingent upon the start or completion of one or more other tasks. A model that is based upon the more generalized framework is able to accommodate the constraints associated with these simplifications as well as ones which are more complex.

It is appropriate to mention at this point that there is a certain duality that exists in the scheduling problem between resources and tasks. The scheduler can allocate resources to tasks or, tasks to resources and it is not uncommon to hear both expressions used within scheduling literature. This duality is normally a characteristic of

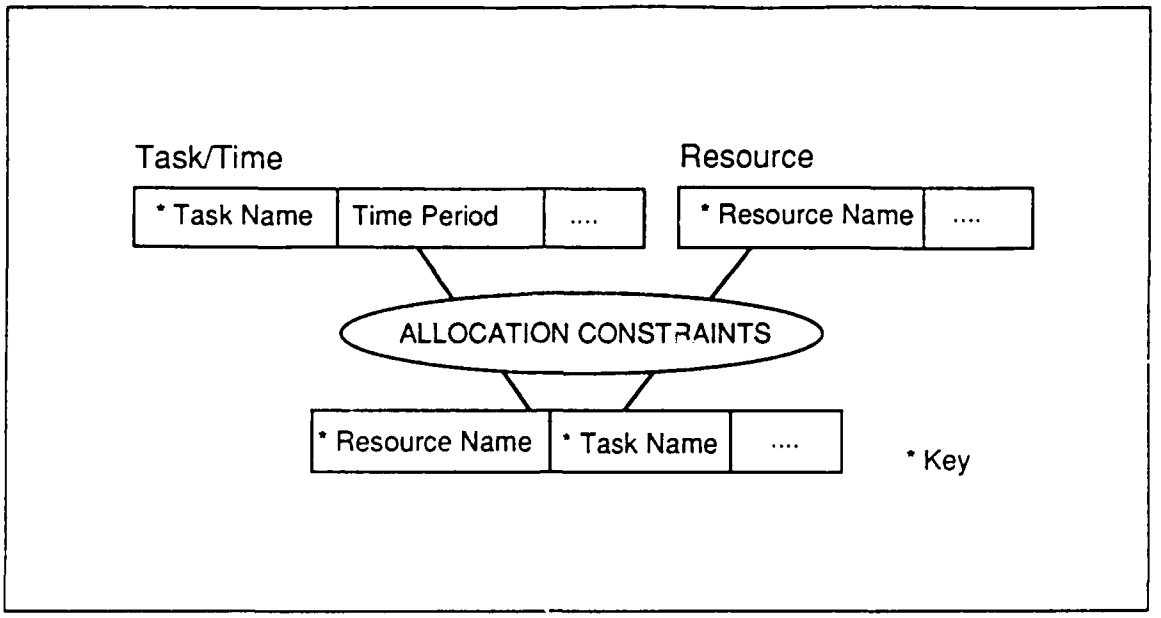


Figure 3. Allocation Constraints

the way the scheduler views the problem and more specifically the perceived scarcity of either the tasks or the resources. The goals and constraints which apply to a specific problem have a direct influence upon the direction of allocation. The direction of the allocation is often translated into a specific solution methodology. The net result, either way, is a schedule which links resources and tasks together in time.

b. Goals

Goals provide the basis for assigning value to feasible schedules for the purpose of identifying a particular schedule which is "optimal". Put another way, the optimal schedule is the one which most effectively addresses the stated goal or goals. The stated goal of most scheduling problems is to minimize total cost, recognizing that resource allocation decisions often have associated costs. These costs are commonly expressed in monetary terms, money being a widely used and uniform measure of relative value. When monetary terms are inappropriate, penalty functions can be incorporated into the objective function which capture the relative effect of failing to achieve the various goals. Regardless of the metric used to evaluate relative value, there is an underlying assumption that all the significant cost factors can be identified and quantified beforehand. This is usually a very difficult task. Baker has identified three types of decision-making goals that are prevalent in scheduling [Ref. 11: p. 5]:

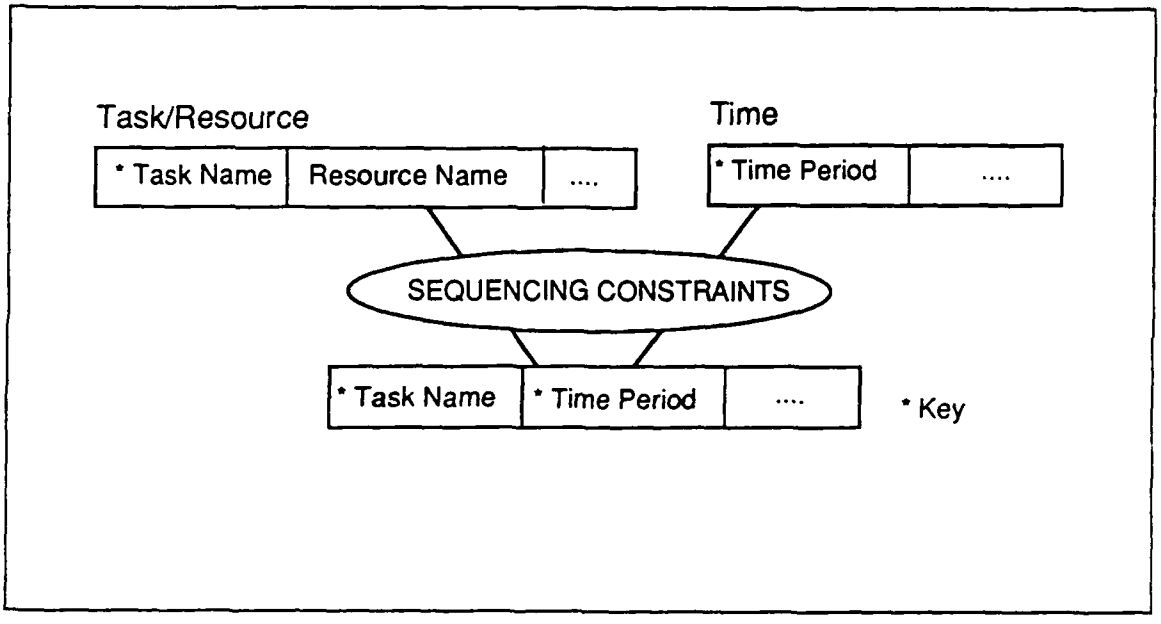


Figure 4. Sequencing Constraints

- Efficient use of resources.
- Rapid response to demands.
- Close conformance to prescribed deadlines.

Special care must be taken to identify scheduling goals which accurately reflect the true nature of the problem. Failure in this regard will result in the production of a schedule which is mathematically optimal but realistically sub-optimal. Here the old axiom of "garbage in, garbage out" is especially true.

C. GENERIC MODEL DEVELOPMENT

The ideal scheduling tool would be one that handles a wide spectrum of problems, containing within it a number of mathematical models. Such a tool would allow the user to enter the unique characteristics of a specific problem into a generic scheduling interface environment or "shell". The system would then select the most effective model or combination of models and provide an optimal solution to the problem. This approach recognizes the applicability of models to more than one problem domain. This same observation was made by Dolk as evidenced in the following statement:

> If a model has proven useful in a decision situation, then we may assume it may prove useful in similar or recurring situations. Given the high relative cost of building models, it makes more sense to modify existing models, if available, than

to start from scratch. Equally important, models may have transfer value so that a model developed in one context may be applicable to different situations as well. [Ref. 13: p. 38]

The process of combining models and providing the intelligence to select the appropriate one (or combination) is a daunting task. The question is, "how can we pursue this eventual goal in smaller, more practical steps?" The answer may exist partially within the conceptual framework which has been developed. Within that framework two separate aspects were identified; the environment and the decision. The environment, as defined, is generic and is applicable to a wide variety of problems. Further, the environment is generally model-independent. During the development of the environment it was pointed out how the two most widely accepted classes of models are accommodated within this framework. It is within the decision area that a system becomes model specific, for certain models have been developed to model certain "decision factors" related to scheduling. The suggested approach to use in the development of these decision factors is one which is generic and modular. Using this approach, a particular solution can be analyzed to identify a set of generic decision factors which it addresses. This approach is similar to the "Problem-Independent Scheduling Systems" paper presented by Loyola, Reilly, and Werntz to the 1989 meeting of ORSA/TIMS:

"Traditionally, scheduling systems have been built trying to directly model a problem. But we feel that the development of more generic systems which we will call problem-independent scheduling systems, can yield greater success." [Ref. 8: p. 1]

If success of a model is based upon widespread acceptance and usage, the greatest successes in OR scheduling models are the Critical Path Method (CPM) and Planned Evaluation and Review Technique (PERT) scheduling models. These two combined top the list of accepted OR models. These are generic scheduling models which are capable of accommodating a wide variety of problems with the specific characteristic that they model. The goal in the remainder of this chapter will be to develop another generic or problem-independent scheduler. Rather than start from scratch, a customized schedule solution model will be explored and then adapted into a generic scheduling model. The model that is selected must be adaptable to a significantly large class of problems.

D. A SPECIFIC OPTIMIZATION MODEL

The approach used in this paper to arrive at a generic scheduling model is much the same as that used by Loyola et al. First a particular problem was selected, and then analyzed with the intention of developing a class of generic problems. It is important

to note that "building a system which would effectively solve all scheduling problems (complete independence) is presently infeasible" [Ref. 8: p. 2]. However, using the framework developed in the earlier part of this chapter, a specific problem may be selected which addresses a broad range of problems. In order to fully exploit that framework a problem which addresses all three components; resources, tasks and time, was selected.

1. The Ship Scheduling Problem

The particular problem solution is one modeled by Wing [Ref. 10]. It involves inter-deployment scheduling of Navy surface combatants during a 13 week quarter. The name given to this computerized solution model is SURFSKED, and that is how this thesis will refer to it as well. SURFSKED was constructed to demonstrate the feasibility of applying optimization methods to solving this complex, hitherto manually solved, problem. The particular methodology involves mixed-integer linear programming and set-partitioning to generate an optimal solution. The model proceeds through three basic steps.

The first step is to generate a set of "candidate" schedules for each ship. This is done using a "generator" which consists of a Fortran program which enumerates only those schedules which meet certain user specified criteria. These criteria are essentially constraints such as those discussed in the conceptual framework. The object of this step is to eliminate from consideration those schedules which are theoretically possible but impractical in this specific scheduling environment. Rather than generating all possible schedules and allowing the linear program to eliminate those which are infeasible, this procedure uses the constraints to generate only those schedules which are feasible for input to the linear programming solver. To distinguish these constraining factors from the constraints used within the linear program itself, the term "filter" will be used when referencing them. The following are the particular filters which are used by SURFSKED to determine whether a schedule for a particular ship is feasible:

- Filter 1. A task must be needed by the particular ship.
- Filter 2. All the prerequisites for a task must be completed by a ship before that task can be scheduled for that ship.
- Filter 3. Tasks may only be scheduled for a ship concurrently with compatible tasks.
- Filter 4. Tasks may only be retaken by the same ship after a specified period.

- Filter 5. The user may directly assign or "lock in" tasks to the schedule, and every "lock-in" task must be included in every attainable schedule.
- Filter 6. A task may not be scheduled during a week when there are no task supply assets available.

The second step assigns to each of the attainable schedules a cost. In the SURFSKED solution this cost is an aggregation of several subordinate *cost factors* which each increase exponentially as they deviate from established ideals. The costs factors which are used to assign cost to each of the candidate schedules are:

- Inter-event sequencing - This factor assigns cost penalties to schedules with separation times between tasks and their prerequisites which vary from the ideal.
- Readiness - This factor assigns a cost to the scheduling of a task based upon its deviation from the ideal periodicity and based upon the ship's priority and the task's importance.
- Tempo - This factor imposes cost penalties for deviation from established ratios (PERSTEMPO and OPSTEMPO) of "in-port" and "at-sea" tasks.
- Deletion - This factor assigns cost penalties to schedules which do not include needed tasks based upon the importance of the deleted task and the priority of the ship.

The final step uses an "optimizer" to determine the combination of schedules which has the lowest cost. "Once all candidate schedules have been generated and their costs evaluated, the solution to the scheduling problem is to select exactly one schedule for each ship such that the set of selected schedules minimizes total costs without violating the supply constraints of supporting (task) assets" [Ref. 10: p. 44]. The specific optimizer used by SURFSKED is one developed by Brown and Graves called the X-System Solver which produces efficient solutions to large-scale integer linear programming problems. The two types of constraints that are evaluated by this linear program are:

- Supply Constraints - The supply of task assets available during each week of the quarter can not be exceeded.
- Set-Partitioning Constraint - Exactly one schedule must be selected for each ship.

The objective function selects an optimal solution by minimizing the total cost of the aggregate schedule. The optimal solution does not necessarily (and in fact probably does not) include the lowest cost schedule for each individual ship. Rather, it identifies the lowest cost aggregate schedule.

To summarize, there are three different categories of factors used by this model to identify an optimal schedule.

- Filters
- Costs
- Constraints

2. The Underlying Mathematical Model

The particular mathematical model used by SURFSKED is an integer linear program referred to as the **Set-Partitioning Model**. This model has the following general mathematical form:

$$\text{Minimize} \quad \sum_{i=1}^{n} c_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_{ij} x_i = 1 \quad \text{for j} = 1,\ldots,\text{m}$$

$$\text{where} \quad x_i = 0 \text{ or } 1 \quad \text{for all i}$$

$$a_{ij} = 0 \text{ or } 1 \quad \text{for all i and j and,}$$

$$c_i \geq 0 \quad \text{for all } i$$

The model is used in this case to select a set (or "partition") of columns which includes only one schedule for each resource.

E. A GENERIC SCHEDULING MODEL

The next step is to identify generic, broadly applicable features from the specific problem structure and solution methodology presented in the foregoing section. The objective is to define a class of problems of which the SURFSKED problem and all similar problems are members [Ref. 8]. The environment and the specific decision factors that may be modeled are identified and discussed in this section.

The decision factors will be developed in a modular fashion, allowing the user to select those that apply to the particular problem at hand. This modularity will also facilitate the later inclusion of additional factors that increase the scope of application. As additional factors are developed a broader scope of problems may be addressed.

1. The Environment

The first step will be to define further restrictions to the basic three dimensions of time, resources, and tasks. Because these three establish the foundation on which the scheduling problem rests, they must be left purposefully broad in order to avoid unnecessarily limiting the problem class. Additional restrictions will be left to the individual user through the identification of filters, constraints, and goals.

a. Time

The generic problem considers time in discrete blocks or units for the purpose of scheduling. The specific problem used weeks as the units, however, in the generic problem the user will be given the latitude of defining the specific units (ie.,hours, days, weeks, months, etc.). The view of time will be confined to a specific time period or window defined by a starting point, an ending point, and the number of encompassed time units. The generic problem will allow the user to specify this window, perhaps placing an upper limit on the number of time blocks which recognizes computational limitations of the underlying model.

b. Resources

Resources will be considered to be single-valued, discrete, and independently scheduleable units. It is important that each resource constitute an inseparable entity rather than a grouping of separable entities or elements. A combination of items should only be identified as a resource if always scheduled and used together in the particular problem. Provision will be made for combining resources into groups or classes for the purposes of referencing as well identification of common attributes important to the scheduling decision (see "Grouping" below). The generic model will accommodate a variety of resource types in much the same way the particular problem accommodated a variety of ship types. Resources will also be considered to be non-consumable, that is, they are not consumed in the execution of a task to which they are assigned.

c. Tasks

The generic model will accommodate only uninterruptible tasks, or tasks which once begun are completed without interruption. The functionality of interruptible tasks may still be accommodated by separating the task into it's non-interruptible components, classifying each as a separate task, and establishing special prerequisite relationships between them. The generic model will be based upon a static set of tasks, that is, the list of tasks will remain constant throughout the scheduling period [Ref. 11: p. 6]. A particular task may occur during more than one period in the schedule, with different resources or even with the same resource if that is appropriate to the problem.

The functionality is also provided for tasks, like resources, to be grouped for the purpose of referencing or identification of similar characteristics.

d. Attributes

Attributes are characteristics or values associated with resources and tasks that are important to the scheduling decision.

Attributes vary in their degree of complexity. The simplest form of attribute is one that depends only on its parent object. These **simple attributes** have a one-to-one correspondence with the parent, and therefore represent a single value.

Other attributes are dependent upon more than one object and are multi-valued. These **compound attributes** may be thought of as matrices of values with the objects upon which they depend forming the ordinates. An example of a compound attribute from the SURFSKED would be the "need" of particular ships for certain tasks. In that problem these values were contained in a matrix of resource by task dimensions.

The attributes that are necessary to the solution of a problem are dependent upon the particular decision factors (constraints and goals) that are being modeled. Attributes constitute the variables which contain the values used by the various decision factors within the scheduling environment. For this reason, attributes are perhaps most logically associated with the decision factors which require them.

e. Grouping

Within the surface ship scheduling problem as well as many other scheduling problems there is an apparent need to be able to deal with resources and tasks both individually and as groups. This latter capability recognizes that certain sets of resources and tasks may have common attributes or be constrained similarly within the decision process. Very often the factors affecting the scheduling decision are oriented towards groups of tasks and resources. Further, providing an ability to group individual resources enables solution of problems where resources are commonly considered collectively. In the generic problem the user will be allowed to identify a group by the creation of a group name and the assignment of either resources or tasks to that name. Once a group has been defined the group name can be used within the scheduling environment to collectively reference all its members. Further this group referencing may be extended to the schedule output allowing information to be viewed collectively.

Critical to this notion of grouping is the concept of inheritance. Inheritance enables each subordinate member object (resource or task) to inherit, or receive from its parent group, the parent's attributes. Attributes applied to groups of resources are inherited by each of the individual resources in the group. This functionality provides a

timesaving shorthand method for assigning attributes common to a number of resources or tasks. Attributes may be assigned directly or they may be inherited. By allowing the assignment of attributes both directly and through group inheritance users can make attribute assignments in the way that is most convenient and intuitive.

The generic model proposed here will provide the ability to create multiple groups of resources and multiple groups of tasks.

2. Generic Decision Factors

For the sake of simplicity, the generic class, like the ship scheduling problem will be handled deterministically and will not address probabilistic relationships between the various objects or their attributes.

The factors that influence the scheduling decision within the generic problem have been organized into the three classes identified within the specific problem; filters, costs goals, and constraints. Each factor will be briefly discussed and the specific attributes which support that factor will be identified. The particular primitive entities and other attributes on which each of the attributes depend will be indicated within parenthesis immediately following the attribute name.

a. Filters

Task-Resource Need. At the heart of this particular problem class is the assumption that not every resource "needs" every task. This allows the elimination of all pairs of tasks and resources which are not feasible in the given problem. This filter maps each resource to those tasks to which it may be reasonably assigned. Associated with each feasible pairing of task and resource is a duration, or number of time periods required to satisfy the need. Need is often dependent upon time and historical information. Tasks, once completed by a resource are generally not required again, if at all, until a certain period of time has elapsed. For example: A sailor (resource) needs regular physical readiness testing (event). This is a recurring task. However, once a test has been successfully completed another test is not required until a certain minimum time has elapsed. By providing this functionality, the generic model is able to easily accommodate recurring tasks without the user having to entirely reassess the needs of all resources at the beginning of each new scheduling period. This capability therefore provides continuity with previous scheduling periods. The attributes required by this filter are:

- *need* (resource, task) - The duration in time periods required, otherwise empty (or false).
- *last completion* (resource, task) - The date the task was last completed.

- *minimum periodicity* (resource, task) - The minimum number of time periods which must elapse before the task is rescheduled.
- *duration* (resource, task) - The number of time periods it takes for the particular resource to complete the needed task.

Prerequisite Satisfaction. A wide variety of scheduling problems contain rules regarding the sequencing of tasks. The significance of this single factor is evidenced by the CPM and PERT scheduling methods which rely almost entirely on this relationship as a schedule determinant. In those models there is normally a single resource upon which all the tasks and their prerequisite relationships depend. This is commonly the major end item (ship, building, etc.) which is the focus of the entire project. This generic model takes a broader approach, allowing different prerequisite relationships between events to be associated with any or all individual resources. For example: Resource 1 may require Task A before Task B and Resource 2 may require Task B before Task A. The generic model will also borrow from the ship scheduling problem the concepts of a minimum separation between the prerequisite and requisite tasks. This minimum separation allows time for such things as transit or intervening experience. The attributes required by this filter are:

- *prerequisites* (need, task) - a list of immediate predecessor tasks for each task-resource pair.
- *last completion* (same as above) - here this attribute is used to identify the prior completion of prerequisites.
- *minimum task separation* (event, prerequisites) - the minimum number of periods that must separate one task from another.

Task Compatibility. This filter is predicated upon the notion that certain required tasks can be performed concurrently (ie., walking and chewing gum). For the sake of simplicity our generic model will limit the number of tasks that can be performed concurrently to three (this is also the maximum that SURFSKED addressed). It should be noted that this capability should only be used when absolutely necessary because of the potential for vastly expanding the scope of the scheduling decision. This filter requires the following attributes:

- *compatibility* (task, need) - a matrix of tasks which may be reasonably performed concurrently. Initially all tasks have a default incompatibility with all other tasks.

Lock-Ins. This factor is based upon allowing the user to make direct assignments of resource time combinations to specific time-frames. This capability is necessary in any environment where the user may occasionally desire to override the

scheduling mechanism to guarantee that a particular task occurs at a particular time. This filter excludes from consideration any schedule which does not include these "locked-in" tasks. The following attributes support this filter:

- *assignments* (resource, task, time) - a matrix of direct assignments which have been entered by the user. Default is no assignments.
- *need* (same as above) - The need matrix is used as a validation check on each direct assignment, to verify both the requirement and the duration. The user should be allowed to override this validation.
- *duration* (same as above) - The assignments must show the appropriate duration for the task.

Resource Availability. Each resource may or may not be available for scheduling during the entire scheduling period. This filter allows the user to indicate periods of availability for each resource. The default assumption is that all resources are available for the entire period. The following attribute is required to support this filter:

- *availability* (resource, time) - this matrix identifies the specific periods of non-availability for each resource.

Task Availability. There may be time blocks when certain tasks may be not scheduled. Only schedules that do not include those task-time combinations are feasible. This filter acts as precursor to the Supply Constraint which will be discussed later. The attributes which are required are:

- *task supply* (task, time) - this matrix identifies the maximum number of tasks that may be scheduled during each time period. This particular filter is concerned with those time periods where that value is zero.

b. Cost Factors

Task Inclusion. This cost factor addresses the value associated with each schedule based upon the tasks that it includes, the importance of each of those tasks and how closely the scheduling of those tasks conforms to the established ideal periodicity for each of the tasks. A lower cost is also given to schedules which involve higher priority resources. The following attributes have a bearing upon this factor:

- *task importance* (need) This attribute assigns to each needed task a value associated with it's relative importance when compared with other needed tasks.
- *resource priority* (resource) - this is a simple attribute that allows the user to assign to each resource a value corresponding to the relative priority it receives in the scheduling problem.
- *ideal periodicity* (resource, task) - The ideal number of time periods which should elapse before the task is rescheduled.

- *periodicity slack* (event) - the number of periods that the periodicity can vary from the ideal without any penalty.
- *attainable schedule* (resource, task, time) - this is the set of schedules generated for each individual resource using the filters previously addressed. Each attainable schedule represents one possible ordering of needed tasks for a given resource over the schedule period.

Task Omission. This factor assigns to each schedule a cost based upon the needed tasks which it excludes. This cost is dependent upon the importance of the task which was excluded and upon the priority of the resource being scheduled. The cost is further affected by how close the event is in time to the ideal period for rescheduling. The particular attributes that are required are:

- *task importance* (same as above)
- *resource priority* (same as above)
- *ideal periodicity* (same as above)
- *attainable schedule* (same as above)

Task Sequencing. This cost factor is concerned with the time spacing between a task and its prerequisites in each attainable schedule. Comparison is made with an ideal separation, and higher costs are assigned based upon variance from that ideal. The following attributes are utilized:

- *prerequisites* (need, task) - a list of immediate predecessor tasks for each task-resource pair.
- *last completion* (same as above) - here this attribute is used to identify the prior completion of prerequisites.
- *ideal task separation* (event) - the ideal number of periods separate a task from its prerequisite.
- *sequencing slack* (event) - the number of periods that the separation can vary from the ideal without any penalty.
- *attainable schedule* (same as above)

Ratios of Task Types. It is often desirable in scheduling to produce schedules which provide a "balance" of various activities. This was true in the ship scheduling problem and is likely true in other environments, too. This particular cost factor allows the user to identify ideal ratios of certain task groups and then to assign greater costs to those schedules which deviate from the ideal. The following attributes are required:

- *ideal ratio* - this attribute stores a user-specified ideal ratio of time spent performing a group of tasks, to total schedule time, for application to all resources.

- *task group* (task) - this is a simple attribute of task which identifies the group or groups of task types that each task is a member of. (See "Grouping" above)
- *attainable schedule* (same as above)

Each of the above cost factors has associated with it certain *weighting factors* which allow it to be "fine tuned" in relation to the other three factors. These weight factors allow users to assign a variable cost profile to each of the factors depending on the specific concerns of their environment.

c. Constraints

One Schedule per Resource. This particular constraint is integral to the particular solution methodology use by the underlying mathematical model.

Supply of Tasks. The generic model allows the user to identify the maximum number of tasks that may be scheduled during each time period to support the combined requirements of all the resources. These constraints represent the only user specified factors which actually constrain the solution.

- *task supply* (same as above) - The maximums specified cannot be exceeded.

F. SUMMARY

This chapter has defined a generic scheduling decision model which includes an environment and a set of decision factors (filters, cost factors, and constraints) used to generate an optimal schedule solution. The attributes that are required to support the model have also been identified. The next task is to design an interface which will accept this information directly from the user. Before that is accomplished, it is necessary to first develop some good interface design principles to be used in the construction of that interface. This will be the objective of Chapter III.

III. INTERFACE DESIGN

A. GENERAL

The proliferation of computers within the 1980's was accompanied by dramatic changes in the way that users viewed and operated those computers. This area of interaction between the computer hardware and the human operator is commonly known as the "user interface" or just "interface". Certainly some of the change in the look of user interfaces may be attributed to a change in the market for computers created by the availability of affordable, personal computers. The interface became an important selling point for many users. Evidence of the importance that computer manufacturers placed on interfaces is seen in the "look and feel" lawsuits which occurred at the end of the decade.

Traditionally, the study of the physical interaction between man and machine has been explored within a discipline called *human factors*. Human factors is described as, "...a discipline which seeks to provide a method for taking into account human strengths and limitations during the design of computer hardware and software" [Ref. 14: p. 108]. More recently these studies have been expanded to include analysis of the cognitive and mental aspects within the growing discipline known as *Human-Computer Interaction (HCI)* [Ref. 14 and 1]. HCI brings together concepts from computer science, psychology, linguistics, anthropology, and sociology to study the way that humans relate to computers [Ref. 15: p.1].

This chapter will explore a number of the recent developments in the design of user interfaces that have come from these two related disciplines. The emphasis will be on general principles rather than on the actual mechanics of interface construction. Recognizing that the objective of the interface is to facilitate smooth communication or dialog between the man and the machine, the goal will be to identify methodologies which make the computer interface more effective, or "user-friendly". The following list of quantitative measures based on work by Shneiderman is helpful in identifying and comparing user-friendliness [Ref. 16 and 17]:

- Less time required to learn
- Less time required to use once learned
- Lower frequency of errors

- Increased user satisfaction
- Improved retention over time

With this list in view, the various aspects of computer dialog will now be explored for principles to guide the development of the user interface for the scheduling DSS application.

B. THE DSS VIEW OF DIALOG

As was mentioned in the first chapter, one of the strengths of the DSS approach is its emphasis on the user interface. For that reason it is useful to understand the view taken by those within the DSS community regarding the user interface, or "dialog", as it is frequently referred to in that literature. A widely accepted conceptual model of the DSS user interface is that developed by Bennett. Bennett provides the schematic representation of the user interface shown in Figure 5. and proposes three distinct aspects of the dialog experience [Ref. 18: pp. 46-47]:

- Action Language - What the user can do.
- Presentation Language - What the user sees.
- Knowledge Base - What the user must know.

The design of every interface makes certain assumptions with regard to each of these. There is a spectrum of assumptions and related designs that will work, but certain methods are more effective than others. Each of these three components of the dialog corresponds to a different aspect of human mental processing. These three systems and their functions are [From Ref. 19]:

- The Motor System - Physical actions are produced through the actions of the muscles moving parts of the body.
- The Perceptual System - External stimuli are detected and transmitted to the cognitive and motor system for further processing or action
- The Cognitive System - The perceptual stimuli, stored knowledge, and stored procedures for responding to the stimuli based on the knowledge are processed to produce appropriate actions. either extra stored knowledge or procedures, and physical actions (using the motor system).

In the following three sections each of these aspects of the dialog is analyzed for principles that have proven effective when applied to the design of the user interface. Because these three are so closely related, there will necessarily be some overlap between the different areas. An effort has been made to discuss each feature in the section where it appears to have the most significant impact.

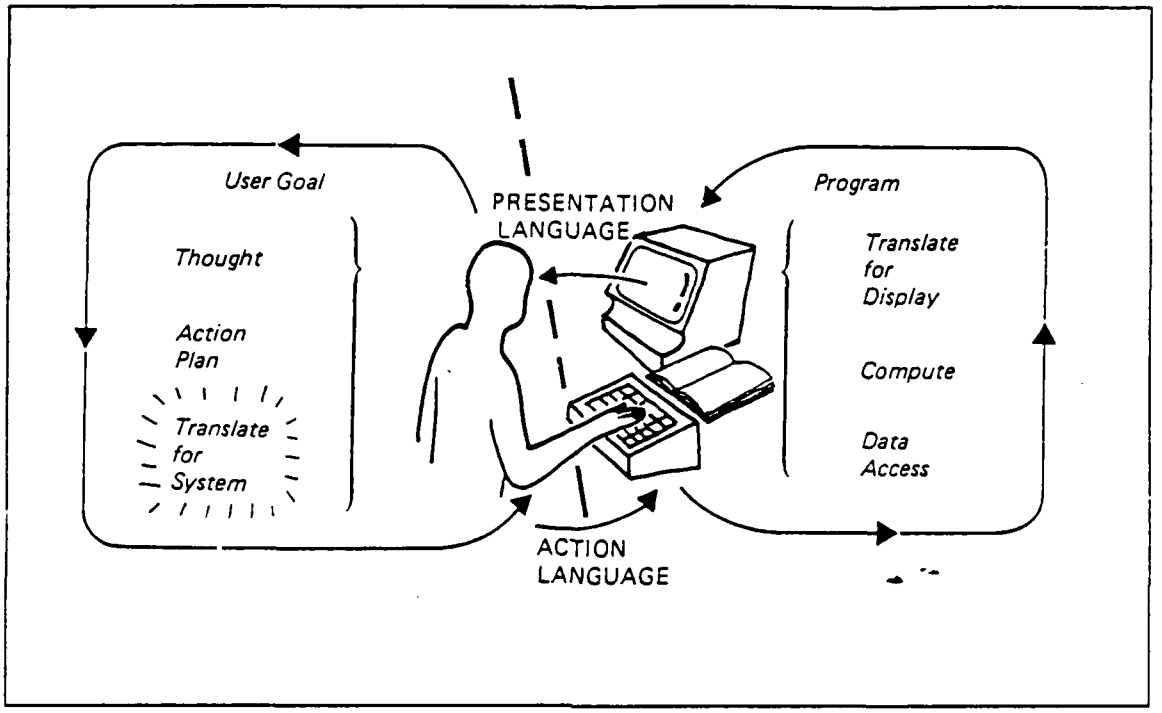


Figure 5. Schematic Representation of a User Interface
Source: [Ref. 18 : p. 45]

1. Action Language

a. Input Devices

The action language corresponds to the user's motor system and so relates to the transformation of bodily motion into meaningful computerized processes. Although new and exciting means of communicating with computers are currently being explored and developed (voice, head and eye movement, etc...), most of the current interfaces are operated by hand and finger movements. This discussion will be limited to that type of input. There are two principal ways of communicating with computers which are in widespread use: keyboards and pointing devices. This later category includes mice, lightpens, touch screens, and trackballs. These input devices have proven to be effective tools whose use is learned relatively quickly by non-computer professionals. They are also widely supported by a number of personal computer systems.

b. The "Production Paradox"

One of the first questions most users have regarding a system is, "what can I do with it?". This predilection to action is the one of the topics addressed in an article

by Carroll and Rosson [Ref. 20] in which they describe a "Production Paradox". This paradox represents an overwhelming desire by users to get their hands on the program and do something with it.

> New users tend to jump right in when introduced to application systems. If an operation is referred to in their training materials, they want to try it out at once. Rote descriptions and practice are resisted, and even when complied with, prove difficult to follow and assimilate. [Ref. 20: p. 83]

The consequences of this approach in many applications can be devastating. It is amazing what damage can be done to many programs by the curious and inexperienced user.

The effects of this paradox are not limited to new users, but, the effect on experienced users is somewhat different. Here the tendency is to develop a method of operation which satisfices rather than optimizes the capabilities of the software. Rather than discovering and using designed shortcuts, users tend to find a way that works and that they are comfortable with, and then continue to use it.

How should the recognition of this "Production Paradox" influence the design of application software? The authors of the article suggest a number of very practical solutions, some of which are outside of the design of the interface itself. One of their approaches to interface design which has seen widespread implementation in recent years is to provide an "undo command" capability. This allows easy reversal of an action which was in ignorance selected with undesired results. This approach can greatly reduce the fear on the part of the new user that they will inadvertently destroy something, thus encouraging them to experiment with the system.

Another approach to this paradox is to try to anticipate user errors during the design process [Ref. 21: p. 582]. This involves second-guessing the types of mistakes that users might commonly make. Trying to second guess all the types of mistakes a user could possibly make is a difficult task, especially if the interface is complex.

An increasingly popular way to prevent the user from taking action with potentially devastating ramifications is to use warning messages. This is very similar in many respects to the undo capability. only it is less voluntary and generally used only to guard very severe actions. Prior to the execution of actions which are very significant, such as erasing files or applications, a warning message will appear which explains in plain language the effect of the action invoked. This provides the user with an opportunity to recover from what otherwise may have been a disaster.

c. Degree of Interaction

Historically, computer system were operated for the most part in a "batch" mode. The user would enter data and command sets and then send it to the central processing unit for computation. Many models today are run in a similar fashion. The trend in recent years has been towards more interactive systems. Interaction involves both the Action Language and the Presentation Language, and the greater the degree of interaction, the shorter the interval between the action and a reciprocal display or feedback. The importance of "feedback" is discussed under Presentation Language. There appears to be a strong preference by users for more interactive systems. One of the benefits of a more interactive interface is that it provides confirmation to the user that the actions were correct, and taken as intended. The negative effects of erroneously applied actions are reduced by making the user aware of them sooner. In the early eighties, Alter studied the benefits of interactive computing and he observed that, "Interactive computing sometimes has a major substantive impact on group planning processes"[Ref. 22: p. 164]. He also found that interactive systems were more valuable when the DSS was being used as a impartial resource used to focus and clarify a position [Ref. 22: p. 163]. These observations support the use of interactive computing for scheduling conferences where the scheduling application is used as a tool in the planning and scheduling activities of the organization.

d. Program Complexity

Another consideration that influences the action language is the overall complexity of the program. Simpson encourages the software designer to "keep the program simple" [Ref. 14: p. 114]. This is not a simple task. Certain activities are by nature complicated, and the solution package must address that complexity if it is to be relevant. Software should strive to be no more complicated than the problem domain that it addresses. Every effort should be made to eliminate that which is unnecessary. The number and type of actions that the user performs for a particular task should be carefully analyzed with the objective of keeping them as simple and intuitive as possible.

Additionally, simple software, although easy to use initially, may be subject to rapid obsolescence as the user becomes more knowledgeable. The ideal software is simple enough to use with little introduction, but powerful enough to meet the needs of the user who has become more proficient with the system and wants to handle increasing complexity, the "power user". This type of functionality is perhaps best demonstrated by an example. Consider the very successful Lotus 1-2-3 spreadsheet program. Basic use of the system is relatively easy. However, underlying the basic command set are

additional capabilities, most notably a macro language, addressed to the "power user". The end result is a software package which is simple enough for a novice to use and yet containing expanded capabilities for the more experienced user. The software accommodates the increasing proficiency of its user.

e. Flexibility

Closely related to the point just made is the notion of multiple paths of action. The concept here is that the software allows the user flexibility in the way the problem is accomplished. The user can do the same task in different ways. Again Lotus 1-2-3 provides a good demonstration of this capability. The user of Lotus 1-2-3 can specify commands in two ways, either by using the cursor to highlight the command or by typing the first letter. User are free to select the method that is most comfortable, giving consideration to skill level and personal preferences. Systems that employ both of the input devices mentioned above can offer the user the alternative of using one or the other to accomplish the same action.

f. Dialog Style

Also integral to the design of the action potion of the dialog is the selection of a dialog style. The most common choices are listed below. The first five are from Sprague and Carlson [Ref. 6: pp. 199-202], and the last two were added by Jones [Ref. 16: p. 18]:

- Question and Answer
- Command Language
- Menu
- Input Form Output Form
- Input-in-Context-of-Output
- Natural Language
- Direct Manipulation

The style of dialog may be any one of these or a combination of two or more of these. There are trade-offs among styles, with each having merit in particular situations and environments. Sprague suggests that most DSSs will combine dialog styles. For example, "direct manipulation" often includes form fill and menu styles. Direct manipulation has seen tremendous growth in recent years and provides a number of significant benefits. Because of its significance and the fact that it tends to affect both the action and presentation languages a later section in this chapter will be devoted to it. One particular combination which has become very popular in recent years is menus and command

languages. This combination accommodates both the infrequent and regular users, and is used in a number of current user interfaces were the user is given the choice of either selecting the action from a menu using the cursor, or typing a command key or function key from the keyboard.

g. Consistency

Consistency within the design is a concept which is applicable to each of the aspects. The user expects the same action to produce the same result. It is likewise important that similar functions are performed in the same manner throughout the program. This has the advantage of enabling the user to apply experience gained in one part of the program to the rest of the program, thus simplifying the learning process. [Refs. 14: p. 112 and 1: p. 20]

h. Maintain User Orientation

The objective here is to help prevent the user from "getting lost" within the context of the program. This is also related to the presentation language aspect of the dialog. It is important to provide the user with sign-posts that tell him where he is, where he can go from here, and how he can get back to where he came from. Menu-driven systems again have an advantage in this respect if properly applied. [Refs. 14: p. 114 and 1: p. 21]

2. Presentation Language

a. Output Devices

The presentation language is directly related to the perceptual system of the human mind. There are various ways that humans perceive information including, vision, hearing, taste, smell, and touch. The computer interface has, to date, relied almost entirely on vision, however, audio interfaces will very likely become more popular in the future. The discussion here will consider only visual output devices.

The two principal output devices used by computers are the display monitor and the printer. In the past what the user saw on the display monitor was constrained considerably by the capabilities of the hardware. In recent years this has changed greatly. The engineers at the Palo Alto Research Center were correct in their 1982 prediction that, "all impressive office systems of the future will have bit mapped displays" [Ref. 23: p. 244]. The quality and clarity of these bit-mapped displays is improving every year, with screens getting larger and resolutions getting finer. The direction of display technology is clearly towards high resolution color monitors.

Printer technology has also improved with increasing use of high resolution laser printers to provide hardcopy textual and graphical output. Gone are the days of the slow dot matrix printers with their characteristically crude output.

The two types of output have also become more closely integrated giving rise to the concept of, "What you see is what you get". Under this concept the user is able to view on the machine everything which will be produced by the printer and reciprocally, produce on the printer anything that is displayed on the screen.

These improvements in display technology provide the software designer with increased flexibility in the presentation of information to the user. The question that naturally arises is "what is the most effective way to present information to the user?" Some of the answers lie outside of the computer industry and within the graphics community. That community has for centuries worked with the presentation of information without the character based restrictions which have only recently been removed from computer displays.

b. Display of Text

Although there are many effective applications for pictures and graphic images, much of the presentation of information still relies upon the use of the printed word or text. An interesting study was performed by Mills and Weldon which explored the readability of text on computer screens. [Ref. 24] The results in many respects are not surprising. They generally show that the more like the standard paper presentation the screen is, the more readable it is. This is to be expected for at least two reasons. First, centuries of experience and trail and error have led to the way information is portrayed on paper media. It is expected that these methods would be very effective otherwise they would certainly have been changed. Second, there is certainly a conditioning effect in operation. The fact that humans learn to read from the printed page as well as spending so much of their lifetimes reading from this media must certainly contribute to the ease with which information portrayed in this manner is understood.

A number of observations were made as a result of this study which should influence the way text is displayed on computer screens. Some of the more relevant observations are [Ref. 24: pp. 352-353]:

- The smaller amount of information on most computer screens as compared with a paper page may reduce reading efficiency.
- An appropriate combination of upper- and lowercase letters seems to be better for continuous reading tasks, whereas words in uppercase seem to be easier to locate when searching a display screen for specific information.

- Words with lowercase letters may be easier to read when the descenders extend below the line of print.
- Variable-width characters may be easier to read than fixed-width characters.
- Larger characters appear better for search tasks whereas smaller characters appear better for reading continuous text.
- More than minimal spacing between text lines appears to improve reading performance.
- High contrast between text and background seems to lead to better performance, with dark characters on light backgrounds being easier to read as long as the refresh rate is rapid (100 cps).

It is clear from reading these conclusions that they support applying lessons learned in the world of paper to the world of the computer display. It is also interesting to note that this experience has been borne out in the marketplace where larger bit-mapped displays which present text in a paper-like fashion are rapidly replacing the original less paper-like character-based displays.

c. Display of Quantitative Data

Much of the presentation language of DSS is concerned with the display of quantitative information. This is especially true of those DSS which are based upon mathematical models. The need to display quantitative information is not new. Man has for centuries performed this task with varying degrees of success. For this reason it is appropriate that in the design of the presentation language of the computer display, the past achievements in the non-computerized presentation of quantitative information be considered. A particularly insightful work on this topic is Tufte's *"The Visual Display of Quantitative Information"* [Ref. 25].

First of all it is important to note that Tufte himself recognizes that a graphical depiction may not always be the most effective way of communicating quantitative or numeric information. He points out that "tables usually outperform graphics in reporting small data sets of 20 numbers or less" [Ref. 25: p.56]. The big benefits of graphics are achieved in the display of larger data sets where tabular data can be overwhelming.

One of the first concepts Tufte explores focuses on what is called **Data-ink**. He defines data-ink as "...the non-erasable core of the graphic, the non-redundant ink arranged in response to variation in the numbers represented"[Ref. 25: p.93]. It is this data-ink which is the essential element of the graphic. He then develops a "data-ink ratio" which is nothing more than the ratio of data-ink to the total ink used to display the graphic. It is the "proportion of the graphics ink which is dedicated to the display

of data-information" [Ref. 25: p. 93]. The goal of graphical displays should be to maximize this ratio. Perhaps the most effective way to do this is to erase non-data and redundant data ink. Non-data ink includes the ink used in labels and grids and other decorations. Redundant ink is that ink which just repeats the same information, such as in bar charts. The designer of graphics should constantly be aware of ways that these two can be removed within reason so that the data and the ink which portrays it are emphasized.

Another point which Tufte makes which is especially applicable to computerized displays of quantitative data is his exhortation to "avoid **chartjunk**" [Ref. 25: p. 93]. There are three basic types of items which are lumped into this category. The first is *unintentional optical art*, which, "...relies on moire effects, in which the design interacts with the physiological tremor of the eye to produce the distracting appearance of vibration and movement". These eye catching fill patterns have become increasingly popular within computerized drawing and statistical programs. The problem with most of these patterns is that they create "noise" which in fact distracts from the data rather than supporting or emphasizing it. The second form of chartjunk mentioned is *the grid.* Very simply, this is overemphasis on the grid framework in which the data is placed. "The grid should be muted or completely suppressed so that its presence is only implicit - lest it compete with the data." [Ref. 25: p. 122]. The third, and final form of chartjunk is *the self-promoting graphic.* This exists when the data plays a secondary role to the presentation format. This form is prevalent in many computerized graphics where often the response is, "Isn't it remarkable that the computer can be programmed to draw like that?" instead of "My, what interesting data." [Ref. 25: p. 120]. Graphics should induce the viewer to think about substance rather than about methodology, graphic design, the technology of graphic production, or something else.

Another important aspect addressed is that of "data integrity". In short, the graphic should avoid distorting the data. Four of the principles suggested to improve graphical integrity are [Ref. 25: p. 77]:

- The representation of numbers, as physically measured on the surface of the graphic itself, should be directly proportional to the numerical quantities represented.
- Clear detailed, and thorough labeling should be used to defeat graphical distortion and ambiguity. Write out explanations of the data on the graphic itself. Label important events in the data.
- Show data variations, not design variations.

- The number of information-carrying (variable) dimensions depicted should not exceed the number of dimensions in the data (ie. Don't use areas to show one-dimensional data)

The final concept of Tufte's involves "**data density**". Data density is defined as the number of separate data entries divided by the area of the graphic. Generally, and within reason, good graphics maximize the data density. That is, they provide a large amount of data in a relatively small space. Obviously there is a limit, but generally, graphics tend to under- rather than over-emphasize data density. One way of applying this approach is to use "small multiples" much like the frames of a movie to display variations in different data sets. This encourages the eye to compare different pieces of data. [Ref. 25: pp. 170-174] Another method is to use "multi-functioning" graphical elements. Perhaps the best example of this method is the stem-and-leaf plot which uses the variable numbers themselves to plot the distribution of values. [Ref. 25: pp. 139-159]

In conclusion, Tufte provides the following five guidelines to be used to evaluate graphical excellence [Ref. 25: p. 51]:

- Graphical excellence is the well-designed presentation of interesting data.
- Graphical excellence consists of complex ideas communicated with clarity, precision, and efficiency.
- Graphical excellence is that which gives the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space.
- Graphical excellence is almost always multivariate.
- Graphical excellence requires telling the truth about the data

d. Feedback

In addition to the presentation of information, the display monitor should also be used to provide feedback to the user. Obviously this is closely related to the action language. Every "action" should ideally have a corresponding "reaction" on the screen. The following quote from Simpson underscores the importance of this concept.

> The user of your program also needs feedback. If he makes a keyboard entry and nothing appears on the screen, then he has no way of knowing that his action has had any effect. In consequence, he may repeat his action or try another, possibly causing something unintended to happen. [Ref. 14: p. 112]

Feedback should also appear on the screen in a place which is obvious and in a manner which is logical to the user. The principles of direct manipulation, which will be explored in depth later, rely heavily on this principle.

e. General Display Techniques

There are a number of very simple yet practical tips that apply to the display of information on a computer screen [Refs. 14 and 1]:

- Access screens by paging, not scrolling.
- Title all screen displays, preferably centered at the top of the screen.
- Center screen displays, that's where the user normally looks.
- Allocate specific screen areas for each type or grouping of information and use these areas consistently.
- Distinctly separate each area of the screen with mechanisms such as blank rows or columns, lines, or color coding.
- Keep screens simple and uncluttered through the use of "white space".
- Follow prevailing conventions--present information from left to right, top to bottom, left justifying text and right justify numbers, aligning them on the decimal point.
- Display information in a recognizable order--for example, alphabetically, numerically, or chronologically.
- Break up long strings of data into independently recognizable units. For example, using a hyphen in telephone numbers.

3. Knowledge Base

In considering the knowledge base, the education and experience of the users must be considered. Each user brings to the system a somewhat different domain of knowledge. The more specialized the target audience of the system, the more critical is this concept of "knowledge domain". The term "knowledge domain" is one which was coined by Brooks in his discussion of difficulties associated with computer programming. [Ref. 26: p. 125] In that discussion he uses an illustration of a linear programming computer model to demonstrate the numbers and types of domains that can be involved in the solution of a particular problem. The goal of the user-interface should be to build as much as possible upon the knowledge already existing with the user. The system doesn't have to be limited to this domain but it certainly should accommodate it. The user should be able to begin with little more than what he already knows and be educated in that which he does not know.

a. Minimize Human Memory Demands

One of the principle goals of the interface should be to minimize human memory demands. The computer does a much better job of storing information than humans who tend to forget things, especially when they are cryptic and infrequently used. Also, it is easy for humans to remember information incompletely or inaccurately.

whereas computers are very good at remembering information verbatim. The bottom line in Simpson's words is to "...rely on computer memory as much as possible". [Ref. 14: p. 114]

One area where effort must be made to ease the memory demands relates to the action language. The user should not be required to remember a large set of obscure commands in order to operate the system. A much preferred way is to provide a menu system where the user can select from a choice of commands. For ease of use by the novice or where the system is used infrequently it is generally agreed that the menu system is more effective.

The other method of easing the memory demands of computer systems pertains to the presentation. Each knowledge domain generally has an associated set of "representations" which are familiar to the users. Sprague's popular ROMC approach to DSS design encourages the designer to focus on the representations used in the decision making process [Ref. 6]. It is important to recognize that many representations only have meaning within a specific user context or domain. Most of the problems traditionally solved using MS OR methods span several knowledge domains, each with different associated representations [Ref. 5: p. 891]. As mentioned in the introduction, the computer interfaces to MS OR solution software historically focused on the domain of the MS OR specialist, incorporating representations that were meaningful to that audience. It was then the responsibility of the MS/OR specialist to convert these representations into others which could be understood by the decision maker within his knowledge domain. The system proposed by this thesis envisions a different end user, and therefore a different knowledge domain involving different representations. The interface can help to reduce memory demands by incorporating representations with meaning in the knowledge domain with which the intended user group is familiar.

b. The "Assimilation Paradox"

Another paradox identified by Carroll and Rosson is that, "...users apply prior knowledge even when it doesn't apply" [Ref. 20: p. 102]. People naturally bring to the computer environment a wealth of experiences which they readily apply to the computer. The results of this are seen in the interpretation of English commands and meanings attributed to graphical images. There is clearly no absolute way of preventing the user from misinterpreting the system, but, there are some approaches that can help. One method suggested is to try and repress any association the user might make with prior knowledge. This can be counterproductive, missing an opportunity to build on the user's existing knowledge base and resulting in increased learning time.

Perhaps the better way of approaching this problem is to exploit the user's existing knowledge by using metaphors which are obvious and true. A good example of this approach is seen in direct manipulation interfaces which will be discussed later. Even this approach can fail, for there are no perfect metaphors. With thoughtful design, however, this approach can be very successful. In implementing this approach, care must be taken in the design of the initial metaphors, ensuring that they are both intuitive and applicable. This concept will be visited again in the discussion of the STAR interface.

c. Online Help

Although the design of the dialog seeks to minimize memory demands and tailor the system to the knowledge domain and specific capabilities of the user group, no system can be entirely understood by all the people all of the time. For this reason it is good idea to include an on-screen help facility in the software. This allows the user to quickly answer specific questions regarding the use of the software without having to leave the environment of the display. The best of these systems are "in-context", that is, they recognize where the user is within the software structure and provide answers quickly to questions that relate directly to that context.

C. DIRECT MANIPULATION

The term "direct manipulation" was first used by Shneiderman to describe interfaces designed with the following properties [Ref. 17: p.201]:

- Continuous representation of the object of interest.
- Physical actions or labeled button presses instead of complex syntax.
- Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

The advantages of direct manipulation can perhaps be best explained by considering the following illustration which was used in an article describing Apple's Macintosh computer:

> Imagine driving a car that has no steering wheel, accelerator, brake pedal, turn signal lever, or gear selector. In place of all the familiar controls, you have only a typewriter keyboard.
>
> Anytime you want to turn a corner, change lanes, slow down, speed up, honk your horn, or back up, you have to type a command sequence on the keyboard. Unfortunately, the car can't understand English sentences. Instead, you must hold down a special key with one finger and type some letters and numbers, such as 'S20:TL:A35,' which means, 'slow down to 20, turn left, and accelerate to 35'.

> No doubt you could learn to drive such a car if you had sufficient motivation and determination. But why bother, when so many cars use familiar controls? Most people wouldn't. [Ref. 27: pp.16-21]

Familiarity tends to be a result of interfaces based upon direct manipulation. There is what Laurel refers to as a "first-personess" associated with this type of interface [Ref. 28: pp.76]. This is a grammatical metaphor which conveys the relationship of the user to the system. Previously most computer systems were second-person oriented (example: "place the diskette in drive B"). The use of direct manipulation makes the computer transparent, with the user operating directly on the objects of interest. The user is provided input mechanisms which easily accommodate incremental changes and provide continuous and responsive feedback which can be used to continue, stop, or reverse the operation being performed. It is not difficult to envision the advantages of such an approach. Shneiderman himself has identified the following virtues associated with systems which employ direct manipulation interfaces [Ref. 17: pp. 201-202]:

- Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- Experts can work rapidly to carry out a wide range of tasks, even defining new functions and features.
- Knowledgeable intermittent users can retain operational concepts.
- Error messages are rarely needed.
- Users can see immediately if their actions are furthering their goals, and if not, they simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions are so easily reversible.
- Users gain confidence and mastery because they are the initiators of action, they feel in control, and the system responses are predictable.

Shneiderman has also identified a number of more subjective benefits that he has observed [Ref. 17: p. 180]:

- Users enjoy using the system.
- Users are eager to show the system off to novices.
- Users are inclined to explore more powerful aspects of the system.

Perhaps one of the best examples of the use of direct manipulation can be found within video games. These games provide a clearly displayed field of action and physical actions to manipulate visible objects within that field. Video games rely heavily on an

effective interface to attract and maintain user interest. A number of the principles that they employ have been proven applicable to any interface design. [Ref. 17: pp. 188-190]

D. GRAPHICAL USER INTERFACES (GUI)

A close look at the current trends in operating systems and software applications reveals an increasing acceptance of graphical interfaces as the interface of choice. This is due to a number of factors including improved hardware capabilities and the new types of people using this hardware. In this environment "graphics offer potential for decision makers who can benefit from interaction with computer-generated representations but who are repelled by computer-oriented detail" [Ref. 18: p. 54). This type of interface, popularized by the Apple Macintosh, has come to be referred to as the Graphical User Interface (GUI), and has gained adherents in almost every major personal computer company.

1. General

What constitutes a GUI? There are variations within the current implementations of GUIs but generally they consist of the following elements [Ref. 29: p. 250]:

- A pointing device, typically a mouse
- On-screen menus that can appear or disappear under a pointing device.
- Windows that graphically display what the computer is doing
- Icons that represent files, directories, and so on
- Dialog boxes, buttons, sliders, check boxes, and a plethora of other graphical widgets that let you tell the computer what to do

A current listing of some of the larger GUI participants and the names given to their respective interfacing systems is provided in Table 1.

The basis for most GUIs can be found in the work initially performed by a group of young researchers at the Palo Alto Research Center (PARC). They are generally credited with developing the first commercial GUI-based personal computer. Considering the widespread success of the type of interface, it is beneficial to explore the underlying research upon which this early PARC interface was based. [Ref. 29]

Table 1. CURRENT GRAPHICAL USER INTERFACES

Organization(s)	Product
Apple	Macintosh
Microsoft	Windows
IBM with Microsoft	OS/2 Presentation Manager
Digital Equipment Corp.	DECwindows
Open Software Foundation	Motif
Santa Cruz Operation (SCO)	Open Desktop
Commodore Amiga	Intuition
NeXT Computer	NeXTStep
Digital Research	GEM
Sun Microsystems	Open Look
Hewlett-Packard with Microsoft	Common X Interface
Hewlett-Packard	NewWave

Source: [Ref. 29]

2. The Star Interface

In 1970, Xerox established a research center at Palo Alto to explore technologies that would be important for their planned entry into office business systems. "PARC researchers were fond of the slogan 'The best way to predict the future is to invent it'" [Ref. 30: p. 22]. The amazing thing is that in many ways they did just that. They were ahead of their time, designing systems which stretched the capabilities of the hardware of that day. Much of the pioneering work on personal computer systems can be traced back directly to research performed by these young engineers. Before Shneiderman had published his first paper on the benefits of direct manipulation the group at PARC was coming to similar conclusions in their pursuit of a truly user-oriented interface for personal computers. In April of 1981 they introduced a personal computer called the 8010 Star Information System which employed a GUI. The goals and principles of interface design which are at the root of this system will be the focus of the rest of this section.

a. The Primary Goal and Assumptions

The Star System was designed and targeted towards the business user with the goal of making the computer itself as invisible as possible. The designers assumed

that this user was primarily interested in accomplishing his business tasks and not at all interested in computers. They also assumed a casual, occasional user rather than someone who used the machine all the time. This led to the design goal of making the machine easy to remember in addition to being easy to learn. [Ref. 30: p. 11]

b. Design Approach

The concepts that PARC developed for the interface of the Star were developed in advance of the writing of any software. PARC invested two years and about 30 person-years just investigating visual interface concepts before starting work on the actual product. This was contrary to so many systems which write the software algorithms first and then tack on the user interface later. The designers worked from the user in, rather than from the hardware out. Their investigations led to the identification of a variety of interface concepts and the classification of those concepts into two groups, easy and hard (Table 2).

Table 2. INTERFACE CONCEPTS

Easy	Hard
concrete	abstract
visible	invisible
copying	creating
choosing	filling in
recognizing	generating
editing	programming
interactive	batch

Source: [Ref. 23]

In the development of their interface, they attempted to avoid the difficult or "hard" concepts and instead focus on the easy ones. This solidified into a number of very specific objectives which will be discussed briefly below [Ref. 23].

Familiar user's conceptual model - Star designers recognized that users have a conceptual model which they develop about a system that helps them understand and use the system. They strove to include analogies and metaphors that were already familiar to the user to describe computer functions and components. It was hoped that in so doing they would make the system less alien and abstract, and easier to learn.

Seeing and pointing versus remembering and typing - This approach was based upon the recognition that (1) conscious thought deals with concepts held in short-term memory and (2) the capacity of the short-term memory is limited. Their goal then was to make everything relevant to the task at hand visible on the screen thus relieving the load on short-term memory. This would allow the user to reserve more of this space for actual business activities. An interesting side benefit of this approach was observed:

> A subtle thing happens when everything is visible: *the display becomes reality*. The user model becomes identical with what is on the screen. Objects can be understood purely in terms of their visible characteristics. Actions can be understood in terms of their effects on the screen. This lets users conduct experiments to test, verify, and expand their understanding - the essence of experimental science. [Ref. 23: p. 260]

These are the same benefits that have already been mentioned in connection with direct manipulation.

What you see is what you get - The Star interface was one of the first to recognize this principle of relating the screen to the printed output. The display resolution then (and even now) is not as high as the printed resolution but even still the displayed approximation greatly improves efficiency by eliminating the print commands previously invoked "just to see what it really looks like".

Universal commands - The objective here was to provide basic generic commands that could be used throughout the system. They were helped along in this by the previous identification and graphical depiction of objects. The user could identify a specific object and then apply a more generic command. The result of the use of universal commands was a simpler and more consistent interface.

Consistency - The importance of this feature has already been mentioned, but it is interesting to note the way that it was implemented within the Star design. The design incorporated *paradigms* for operations used throughout the system. It also forced a classification of objects within the system, allowing similar objects to be handled in similar manners.

Simplicity - The designers of the Star attempted to comply with a maxim attributed to Alan Kay: "simple things should be simple; complex things should be possible." Many features already discussed had the effect of enhancing simplicity. One means to this end was to eliminate alternative ways of performing the same function. Also, research was performed with representative users to determine which methods were simplest.

Modeless interaction - Star's designers recognized that confusion often stems from having a variety of modes in which commands are different or have different results. They also recognized that modes were necessary and not all bad. One way to eliminate some of the problems was to use a noun-verb structure to invoke commands. A user would select the object upon which the action was to be performed and then select the action to be performed. They also clearly identified a change in mode, both by a message and a change in the cursor icon.

User tailorability - A final design objective was based upon recognition that no matter how general or powerful an interface is it will never satisfy all users. With this in mind the system was designed to allow the user to change the appearance of the system as well as redefine certain system operations. This allowed the user to adapt the system specifically to his own personal desires and capabilities.

It is clear from the foregoing list of features that the Star embodied a good number of interface principles that have become standard practice within current GUI design. Some of these have been refined in recent years but the basic concepts remain amazingly sound. The designers of the Star interface summarize their efforts as follows:

> User-interface design is still an art, not a science. Many times during the Star design we were amazed at the depth and subtlety of user-interface issues, even supposedly straightforward issues as consistency and simplicity. Often there is no one "right" answer. Much of the time there is no scientific evidence to support one alternative over another, just intuition. Almost always there are trade-offs. [Ref. 23: p. 282]

The designers at PARC did much to move the design of interfaces towards a science. Since the design of the Star, many of the interface principles upon which it was based have been adopted by other manufacturers. Few people would now dispute that they designed the prototype of future interfaces.

E. A LOOK AT TOOLS

It is useful to look at some of the tools that have been recently popularized for the construction of user interfaces. This will be a broad overview rather than a technical examination. Three important technologies will be introduced with the goal of identifying the basic concepts upon which they are based.

1. Object-Oriented Programming (OOP)

The programming world is becoming object-oriented and again this is closely related to what has happened with computer interfaces. Object-oriented programming is based upon basic building blocks called *objects*. Meng provides this definition of an object:

> An object combines information and any operations that can be performed on that information into one single bundle. (in OOP lingo, this bundling is termed *encapsulation*, the operations are *methods*.)
>
> An object performs one of its methods when you send it a *message*. The important thing is that the message doesn't have to tell the object *how* to do something, only *to* do it.
>
> ...Think of objects as computer chips--small modularized units with built-in sets of instructions to perform small operations, ready to be combined into a larger program. [Ref. 31: p. 174]

Hartson adds two additional characteristics not mentioned above; dynamic binding and inheritance of attributes and procedures. He also gives the following advantages of object-oriented programming [Ref. 32]:

- Higher productivity because code can be shared and reused
- Decoupling of representation from implementation
- Reliability, consistency, and locality of definition from inheritance
- Low code bulk

Object-oriented programming was first developed at PARC in the 1970s with the language called Smalltalk. Although the Star interface is not written in Smalltalk many of the concepts learned in the development of Smalltalk were used in development of that system. Today a number of traditional languages including Pascal, C , LISP, and Prolog are being modified to include object-oriented capabilities. [Ref. 31]

2. Toolkits

With the advent of graphical interfaces based on objects, systems of these objects have been developed which are commonly called "toolkits". A toolkit "...provides programming abstractions for building user interfaces" [Ref. 33: p. 19]. These toolkits basically provide a collection of standard objects that can be used to build graphical interfaces. Two of the earliest and best known toolkits were the Smalltalk Model-View-Controller and Apple's MacApp for the MacIntosh. Toolkits are often employed within specific hardware environments for the purpose of maintaining a consistent "look and feel" among the various software applications written for that hardware. Toolkits provide the advantage to programmers of not having to develop objects from scratch. The increasing popularity of both graphical interfaces and object-oriented programming will undoubtedly lead to an increasing use of toolkits. [Ref. 33: p. 19].

3. User-Interface Management Systems (UIMS)

UIMS are a very recent development and it is not always clear in what way they differ from toolkits. The primary difference appears to be in the level of abstraction,

with UIMS generally striving for a higher level of abstraction than toolkits which tend to be implemented within a specific GUI environment. The following explanation of a UIMS has been provided by Norman, Draper and Bannon:

> A UIMS provides a way for the designer to specify the interface in a high-level language. The UIMS then translates that specification into a working interface, managing both the details of the display and the associated input and output and also the interaction with the rest of the program. UIMS systems allow the generation of high-quality interfaces with much less effort than programming the interfaces directly. The disadvantages are that one is restricted to the type of interface supported by the particular UIMS, which may not always match the desired application well. Current UIMS systems are in an early stage of development and the high-level language is not very usable, very complete, or particularly at high-level. In addition there are often penalties in performance. [Ref. 34: p. 496].

In addition to specifying the interface at a high-level of abstraction, UIMS also seek to completely separate the code that implements the user interface from the code that runs the application [Ref. 33: p. 19]. UIMS came about in part due to the criticism that toolkits were too low-level and difficult to work with. However the high-level approach of UIMS has resulted in its own set of shortcomings. The result is that interest in toolkits appears to be on the rise while UIMS have yet to gain widespread acceptance.

F. SUMMARY

This chapter has identified some of the more important principles associated with interface design. The focus has been primarily on graphical interfaces or GUIs because they are undoubtably the interface of the future. The discussion has remained purposely one step behind the leading edge because much of that work remains untried and unproven. Rather the focus has been on principles which, although in some instances quite revolutionary, have been proven valid not only in the laboratory, but also in the marketplace.

It is important to remember that a good user interface is not a panacea. In fact a better interface may be thought of as a two-edged sword in that it will not only make clear what the application can do, but also what the application cannot do. Clearly "form must follow function". With these thoughts in mind, the next chapter will apply the concepts of this chapter to ***TaskMaster***, a prototype interface based on the generic scheduling model developed in Chapter II.

IV. PROTOTYPE DESIGN AND IMPLEMENTATION

A. THE PROTOTYPING APPROACH

Historically, design of computer software was done first on paper. The analyst described in detail the particular features of the system and the entire structure of the program was specified before the actual programming began. In many instances this approach proved time consuming and error-prone. It frequently resulted in serious errors or omissions in design which went undetected until the coding began. [Ref. 21: p. 414]

In recent years there has been a shift towards an engineering approach to design called *prototyping*. The dictionary defines a prototype as, "an original or model on which something is patterned" and or "a first full-scale and usually functional form of a new type or design of a construction (as an airplane)." DSS development is normally performed using an iterative prototyping approach [Ref. 35: p. 152]. Prototyping offers a number of advantages to the design of software which make it an attractive alternative to traditional software design methods.

1. Advantages of Prototyping

The following are some of the advantages of a prototyping methodology [Refs. 21: p. 415-416 and 35: p. 152]:

- Increased and more frequent participation by users increases user acceptance and support for the project and reduces the likelihood of rejection at the time of final product delivery.
- Prototypes are easier for users to understand. If a picture is worth a thousand words, a working model is worth a thousand pictures. This makes it easier for users to provide meaningful feedback.
- Prototyping can result in more creative designs because of better and more frequent user feedback.
- The iterative nature of prototyping allows it to accommodate changing user requirements more easily.
- Prototyping tends to encourage realism in the design, avoiding the error of specifying the impossible or at least making the impossible apparent sooner.
- Prototyping usually results in a shorter overall development time.
- The Prototyping methodology involves lower risk. The feasibility of a project can be reevaluated at the various iterations of the design with a higher degree of accuracy and reliability.

2. Types of Prototypes

Prototyping methodologies may be classified along three orthogonal dimensions, with each dimension embodying two development approaches [Ref. 36: p. 47].

a. Revolutionary (Throwaway) Versus Evolutionary

"A revolutionary development process is one in which a prototype is designed, built, evaluated, and scrapped before work begins anew on the real system" [Ref. 36: p. 47]. Fred Brooks has said, "where a new system or new technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time" [Ref. 37: p. 116]. The very nature of prototyping recognizes this concept and to a certain extent every prototype involves at least portions which are thrown away during the development process.

The other approach in this dimension is evolutionary. Here the prototype, "...evolves through iterative modifications into a complete implementation of the target application system" [Ref. 36: p. 47]. The evolutionary approach minimizes wasted effort and avoids the, "difficult question of when to discard the prototype and start working on the real system" [Ref. 36: p. 47].

b. Interface Only (Rapid) Versus Whole System

The "Interface only" or "Rapid prototyping" approach results in a mock-up of the system which demonstrates the key features of selected components of the entire system rather than the entire system. The focus is generally on the interface component and the ability to demonstrate the "look and feel" of the system to the user.

Using the "Whole system" approach a developer builds the entire system on the computer much like it was built on paper under more traditional development. "Whole System" prototypes are more structured, but are also more difficult to build. [Ref. 36: p. 47 and 21: p. 420]

c. Intermittent Versus Continuous

"Prototypes for which the ability to demonstrate system behavior is 'Intermittent' can be exercised only at times in the development process when a particular version of the system has been completely developed" [Ref. 36: p. 47].

"Continuous" prototypes, in contrast, can be exercised at any time during the development process. This ability is difficult to achieve because it requires that all unfinished portions be appropriately "stubbed" to prevent the system from crashing.

B. NEXT INTERFACE CONCEPTS

A decision was made to develop the interface prototype using the NeXT computer. This decision was based in part on the quality of its bit-mapped display, its computational power, and its interface development environment. This section describes the basic design philosophy of the NeXT interface; its action paradigms; and its interface development application, "Interface Builder". Much of the information for this section was taken from the NeXT System Reference Manual [Ref. 38].

The NeXT user interface was designed to meet the demands of both the novice and the experienced user. To accommodate the novice or infrequent user, the interface should be simple to learn and easy to remember. To meet the needs of the more experienced user, the system should be fast and efficient, without unnecessary or cumbersome mechanisms that slow operation. "The challenge is to accommodate both these goals in ways that don't conflict--to combine simplicity with efficiency" [Ref. 38 p. 2-5]. The graphical, mouse-based, user interface of the NeXT which uses buttons, windows, sliders and other graphical objects with physical counterparts, is able to meet this challenge. Not only are these graphical objects easy to remember they also minimize keystrokes and speed operations, resulting in improved efficiency.

The NeXT interface was designed with the following four basic principles in mind [Ref. 38: p. 2-6]:

- The interface should be consistent across all applications.
- The user is in charge of the workspace and its windows.
- The interface should feel natural to the user.
- The mouse, rather than the keyboard, is the primary instrument for user interaction with the interface.

1. Action Paradigms

The NeXT supports four different action paradigms corresponding to mouse input. Each of these is discussed briefly in the following paragraphs. [Ref. 38: p. 2-6]

a. Direct Manipulation

Most objects respond directly to manipulation with the mouse--a button is highlighted when pressed, a window comes forward when clicked, the knob on a slider moves when dragged. Direct manipulation is the most intuitive of the action paradigms and the one best suited for modifying the position and size of graphical objects. Windows, for example, are reordered, resized, and moved through direct manipulation.

b. Control Action

Some objects--buttons, scrollers, and text fields, among others--are vehicles for the user to give instructions to an application. By manipulating the object, the user controls what the application does. Clicking a "close" button, for example, not only causes the button to become highlighted, it also removes the window from the screen. The button is simply a control device--like a light switch or a steering wheel--that lets the user carry out a certain action. Graphical objects that play this role are therefore collectively known as **controls**.

c. Target Selection

Some controls act on a specific domain. The user first selects what the control should act on, the **target**, then chooses the control. For example, a user might select a range of text in a file, then choose the "Cut" command from the Edit menu to remove it. The selection of a target always precedes the choice of a control action. Selected objects are usually editable graphics or text, but they may also be other types of objects, such as windows (the "Close" command) and icons (the "Delete" command).

d. Tool Selection

In this paradigm, users can change the meaning of subsequent mouse actions by selecting an appropriate tool, often displayed in a palette with several other tools. Each tool controls a certain set of operations that are enabled only after it is chosen. For example, a graphics editor might provide one tool for drawing circles and ovals, another for rectangles, and still another for simple lines. Depending on which tool is chosen, mouse actions (clicking and dragging) will produce very different visual results. The cursor assumes a different shape for each tool, so that it's apparent which one has been selected, and the tool remains highlighted.

Tool selection, in effect, sets up a mode-- a period of time when the user's actions are interpreted in a special way. A mode limits the user's freedom of action to a subset of possible actions, and for that reason is usually avoided. But in the tool-selection paradigm, the mode is mitigated by a number of factors:

- The mode isn't hidden; the altered shape of the cursor and the highlighted state of the tool make it apparent which actions are appropriate.
- The mode isn't unexpected; it's the result of a direct user choice, not the by-product of some other action.
- The way out of the mode (usually clicking in another tool) is apparent and easy. It's available to the user at any time.
- The mode mimics the way things are done in the real world. Artists and workers choose an appropriate tool (whether it's a brush, a hammer, a pen, or a telephone) for the particular task at hand, finish the task, and choose the next tool.

2. Interface Builder

One of the most significant features of the NeXT computer is its "Interface Builder" application. Interface Builder allows the developer to create an interface using

the principles of direct manipulation, by actually "dragging" different graphical interface objects into windows and modifying them as required. It is essentially a "Draw" program for graphical interfaces. The use of Interface Builder makes the construction of a graphical interface (historically a very difficult and time consuming task) a manageable activity. There is no question that this revolutionary approach will become increasingly popular in the future. Without Interface Builder it would have been practically impossible to develop this prototype in such a short period of time. [Ref. 38: p. 2-6]

C. THE SCHEDULING INTERFACE

1. Scope of Implementation

The primary objective of ***TaskMaster*** is to demonstrate the type of interface that can be constructed on a model-based DSS using current graphical interface tools. In this sense, ***TaskMaster*** is a "proof of concept". ***TaskMaster*** currently exists as an "Interface Only" prototype, without connections to the mathematical model. The use of Interface Builder allows the prototype interface to be "continuously" operated throughout the development process. ***TaskMaster*** is also an "evolutionary" prototype which has been designed explicitly for combination with a particular model. The computer platform has been selected in part based upon its ability to handle the computations which are required. The use of an object-oriented methodology will facilitate its evolution into a complete system. The eventual goal is to provide a user-friendly interactive front-end which is fully integrated with the optimizing solver and which facilitates the input of data values and model parameters, eliminating the tedious generation of fixed formatted files and providing more control over the model [Ref. 39: p. 891]. Jones has identified the following steps and associated representations which traditionally accompany the modeling process:

> Consider a problem that will eventually be analyzed using mathematical programming. Tackling that problem will usually involve at the minimum, six different representations. In particular, the problem is usually stated in a natural language (representation 1). Then an algebraic formulation is frequently developed (representation 2). Once relevant data have been collected (representation 3), it must be converted into a form acceptable to the implementation of the mathematical programming algorithm (representation 4). Once the algorithm runs, it typically produces output (representation 5) describing the optimal levels of the decision variables as well as sensitivity analysis information. The output is usually not suitable for managerial presentation, so data from the algorithm output must be extracted to produce a more suitable representation (representation 6). [Ref. 5: p. 891]

TaskMaster has been designed to effectively eliminate the intermediate representations (2 through 5 above), allowing the user to input scheduling information in a natural and intuitive manner and receive meaningful feedback directly.

2. General Principles

In addition to the interface design considerations presented in earlier chapters, the following more specific principles or concepts were developed during the design and implementation of ***TaskMaster***

a. Minimize Data Entry Burden

The objective was to require that the user only put in essential data and that only once. Mathematical models typically involve large matrices of data which are often sparsely filled. The user should only have to fill in the non-zero data. Where the choice of values is limited, the user should be given the opportunity to select a value rather than having to type one in.

b. Provide Maximum Model Information and Control

The data that the model uses and the way that it uses it should be made as clear to the user as possible. The formulas and the basic operations of the model should be available for review and all of the parameters should be adjustable from the interface rather than "hard-coded" into the program.

c. Left to Right, Top to Bottom

Graphical interfaces allow the user considerable freedom of movement within the environment, and a number of actions are available at any time. To guide the user in the proper sequence of actions the ***TaskMaster*** interface employs a left to right, top to bottom paradigm. The selections to be made by the user in the most common sequencing of events should begin at the top or left hand side of the window with subsequent selections below or to the right. This paradigm mirrors the way English text is read and provides a natural ordering to the windows.

d. White for Selection and Input Values

The use of white in the window is used consistently in the interface to indicate that values can be typed into that space or that the highlighted item is being considered. The concept here is that the information over which the user exercises control is highlighted in white. This, along with the ordering paradigm, improves user orientation. Conversely, information which is for "display only" is shown in a dark gray field.

e. Window Specific Help

In implementing the "On-line Help" concept from the Interface Design chapter, a decision was made to include a button in each window that the user could

select to get help. The buttons, indicated by a simple question mark, are the same on all windows and are located generally in the same location (lower right corner). When "pushed" they bring up an information panel which explains the functionality of the window.

f. Save The Visible Information Only

Changes made to data must be either saved or discarded before the user is allowed to exit a window. The user should never be required to make a decision to save or not save information that is not visible. That is asking for errors.

g. Develop and Reuse Standard Interface Objects

This approach creates a consistent interface that minimizes user memory. Within the object-oriented environment of the NeXT this is easy to do and speeds development. For example, within ***TaskMaster*** a standard list entry object has been developed which is used to enter resource, task, and group names.

3. Functional Descriptions

This section briefly describes how ***TaskMaster*** works and describes the functionality of each of the major windows that comprise the application. The figures shown are actual "screen dumps" of individual windows on the NeXT display which allows multiple windows to be displayed simultaneously.

a. Getting Started

When the user first launches the application the menus and title panel shown in Figure 6appear on the screen. Other than the standard NeXT menu choices (Info, Edit, Print, Hide, and Quit) the only choice available (not gray) is "Model". The user selects from the Model sub-menu either a "New" model or "Open(s)..." an existing model. Selecting an existing model loads a previously developed problem into ***TaskMaster***. Creation of a "New" model allows the user to input a different data set. Upon selection of "New" from the menu an empty "Environment" window is automatically opened (see Figure 7). The user gives the new model a name and inputs the time period, resource, and task information corresponding to the scheduling problem to be considered.

b. Scheduling Environment

The "Environment" window, Figure 7, is displayed when "Environment" is selected from the Main Menu (or automatically if a new model is being created). It is designed to allow the user to enter or edit the names of the resources and events in the scheduling decision. The user simply puts the cursor in either one of the white boxes below the scrollable lists and starts entering the names. The list can be edited at any

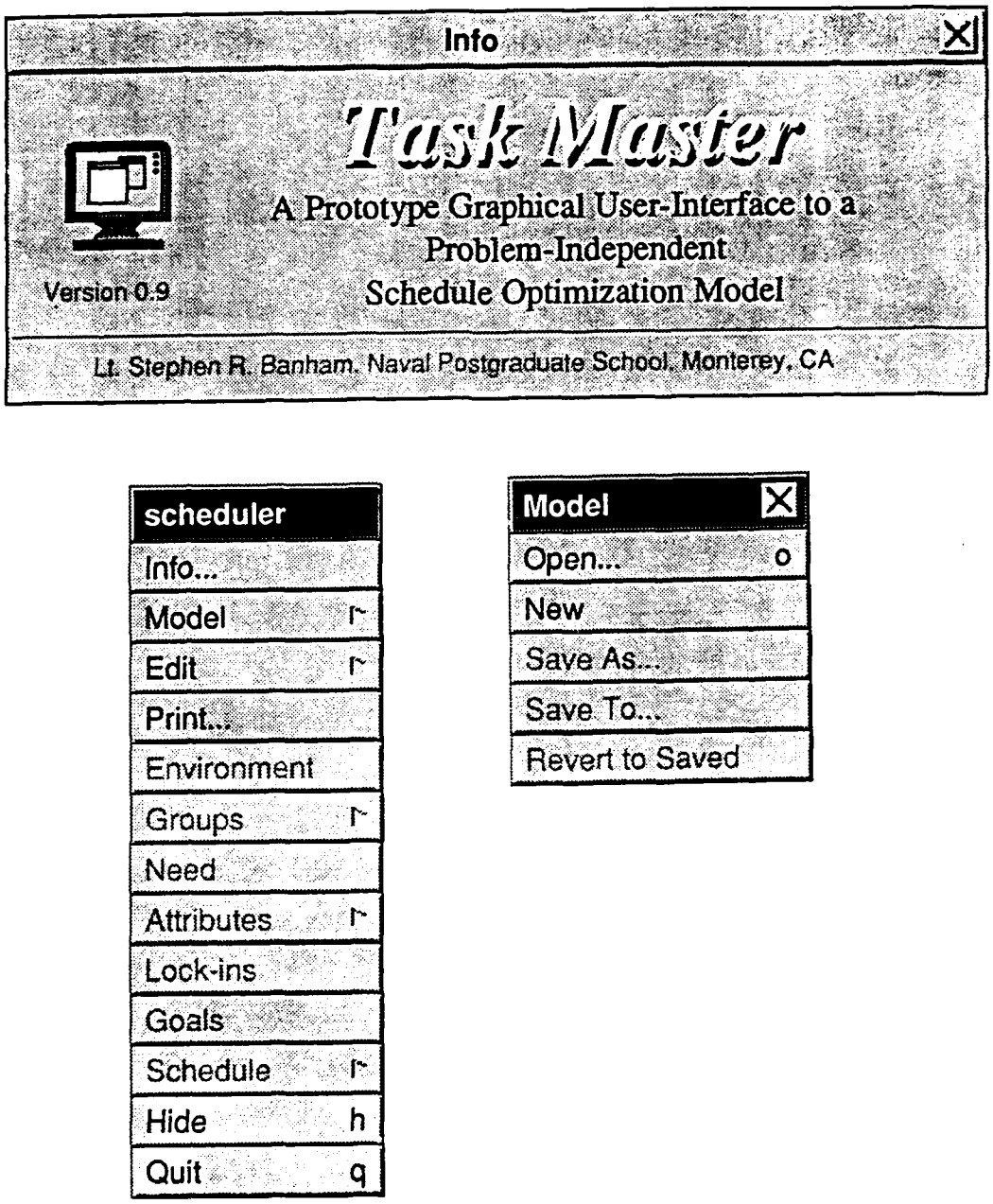


Figure 6. Menus and Title Panel

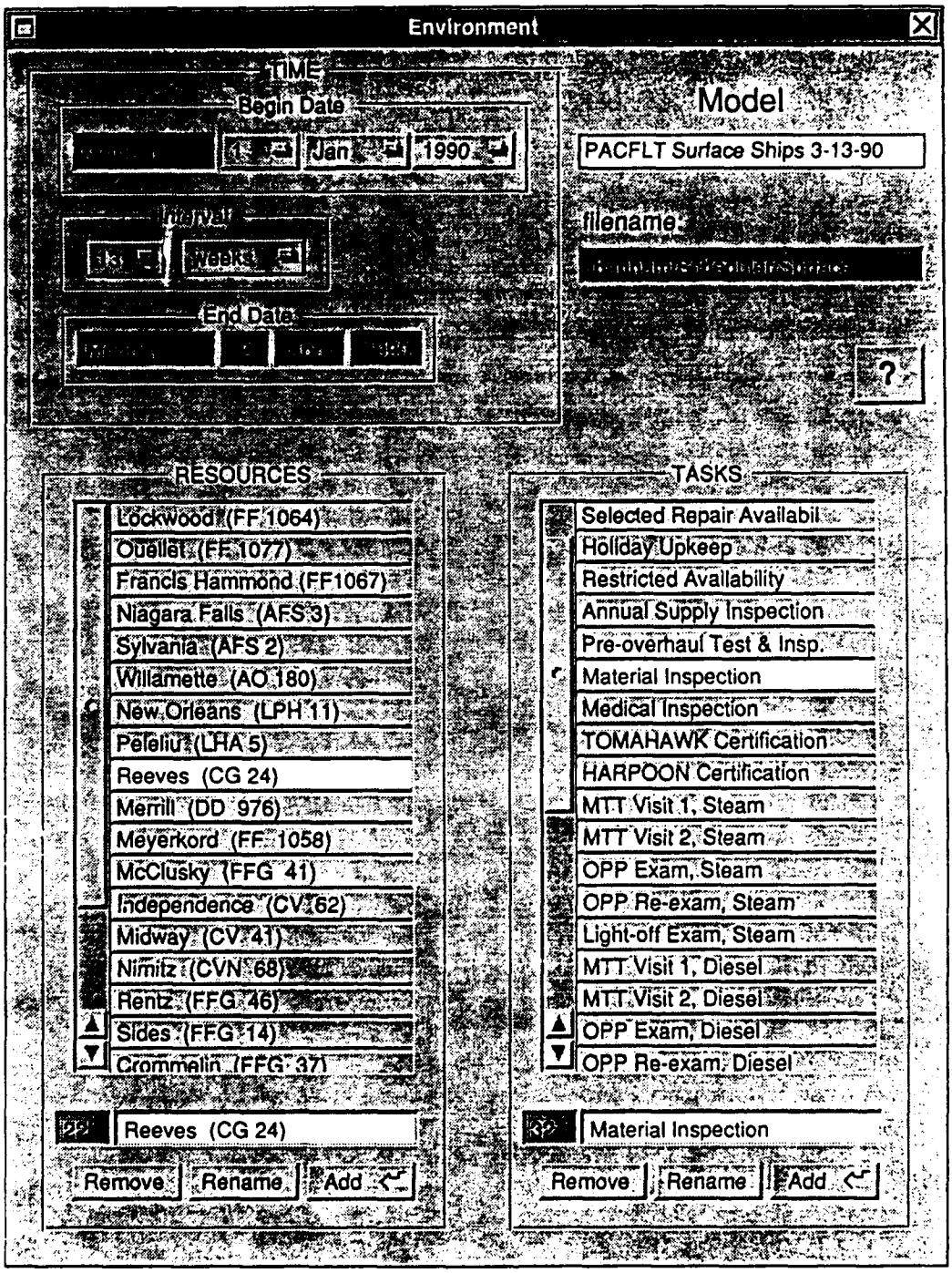


Figure 7. Environment Window

time by selecting the name that needs to be changed (or removed) and then making the necessary modifications using the buttons below the lists.

This window also allows the user to define or modify the timeframe under consideration. A date, month and year is selected from the pop-up lists that appear under the beginning date. The user then selects the number of time periods and unit of time (day, week, or 4 weeks).

Within this window all three "dimensions" of the problem are defined, therefore, this window establishes the size of the model and its underlying database. These are the primitive entities which form the foundation of the model and changes to the values in this window affect all the other windows.

c. Resource Group Assignments

The "Resource Group" window, Figure 8, is displayed when "Resource" is selected from the "Group" Sub-menu. It is designed to permit the user to create group names for resources and assign one or more individual resources to the group. The same scrollable list and input window object is used for creating or modifying group names. To assign member resources to the group, the user simply selects a group (which highlights it) and then selects individual resources from the list to the right (which also highlights them). The assignment is saved by pushing the "Save" button. The "Revert" button allows the user to undo group assignment revision, redisplaying the previously saved assignment.

d. Resource Attribute Assignment

The "Resource Attributes" window, Figure 9, is displayed when "Resource" is selected from the "Attributes" Sub-menu. It enables the user to assign the simple attributes of *Availability* and *Priority* to each resource. This can be done individually, or by group (where all members of the group have exactly the same values for these attributes). When a group is selected, it is highlighted along with all the individual resources that comprise that group. The user then pushes the buttons over each of the time periods in the schedule that the resource is unavailable (the default is "YES", indicating availability for the entire scheduling period). The sliding window mechanism allows the user to scroll through the entire schedule window regardless of its size. The user is also able to select one of five different priorities. Once all attributes have been defined the "Save" button is pushed to store the values. The "Revert " button allows the user to undo attribute changes. redisplaying the previously saved attributes.

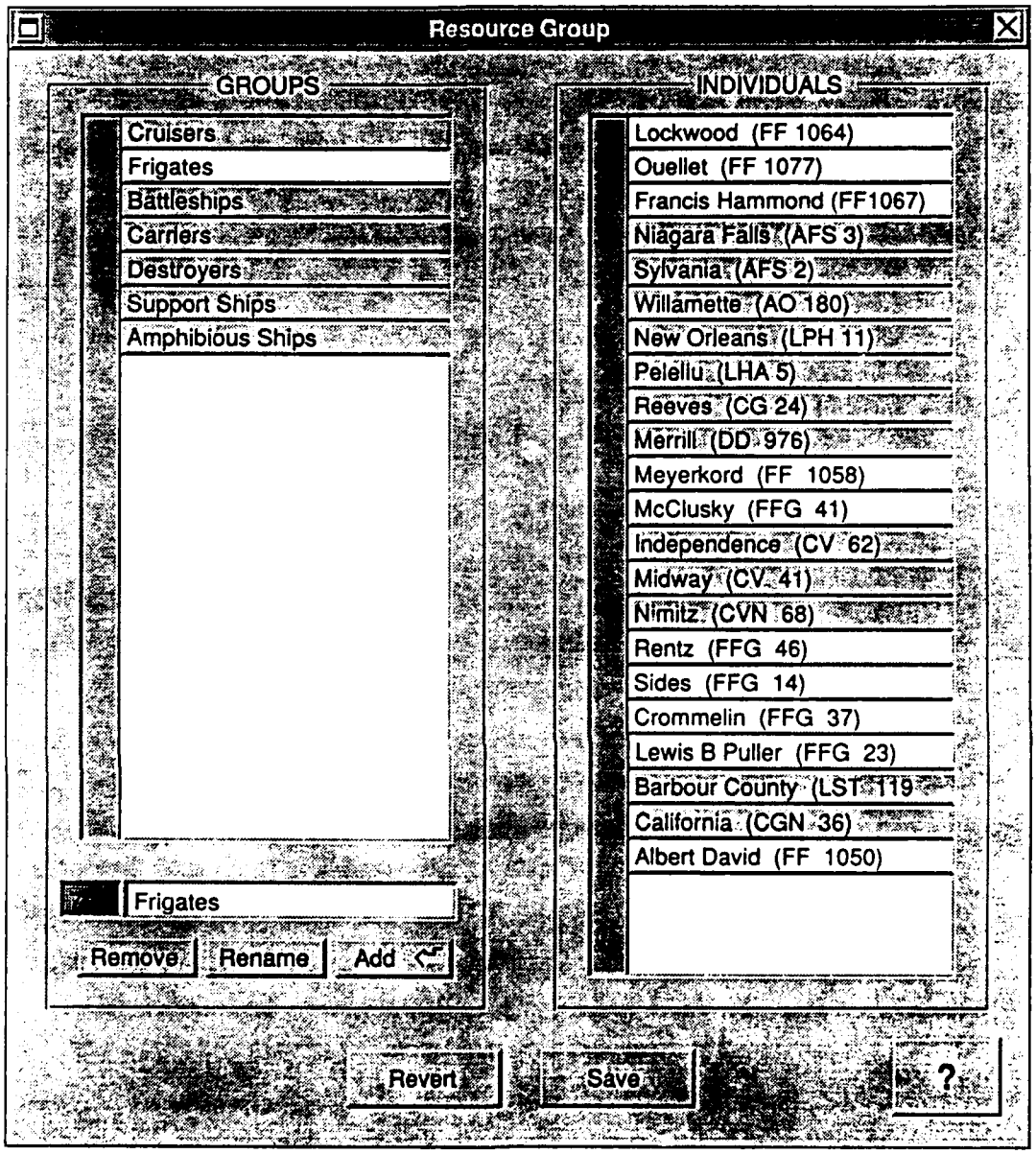


Figure 8. Resource Group Window

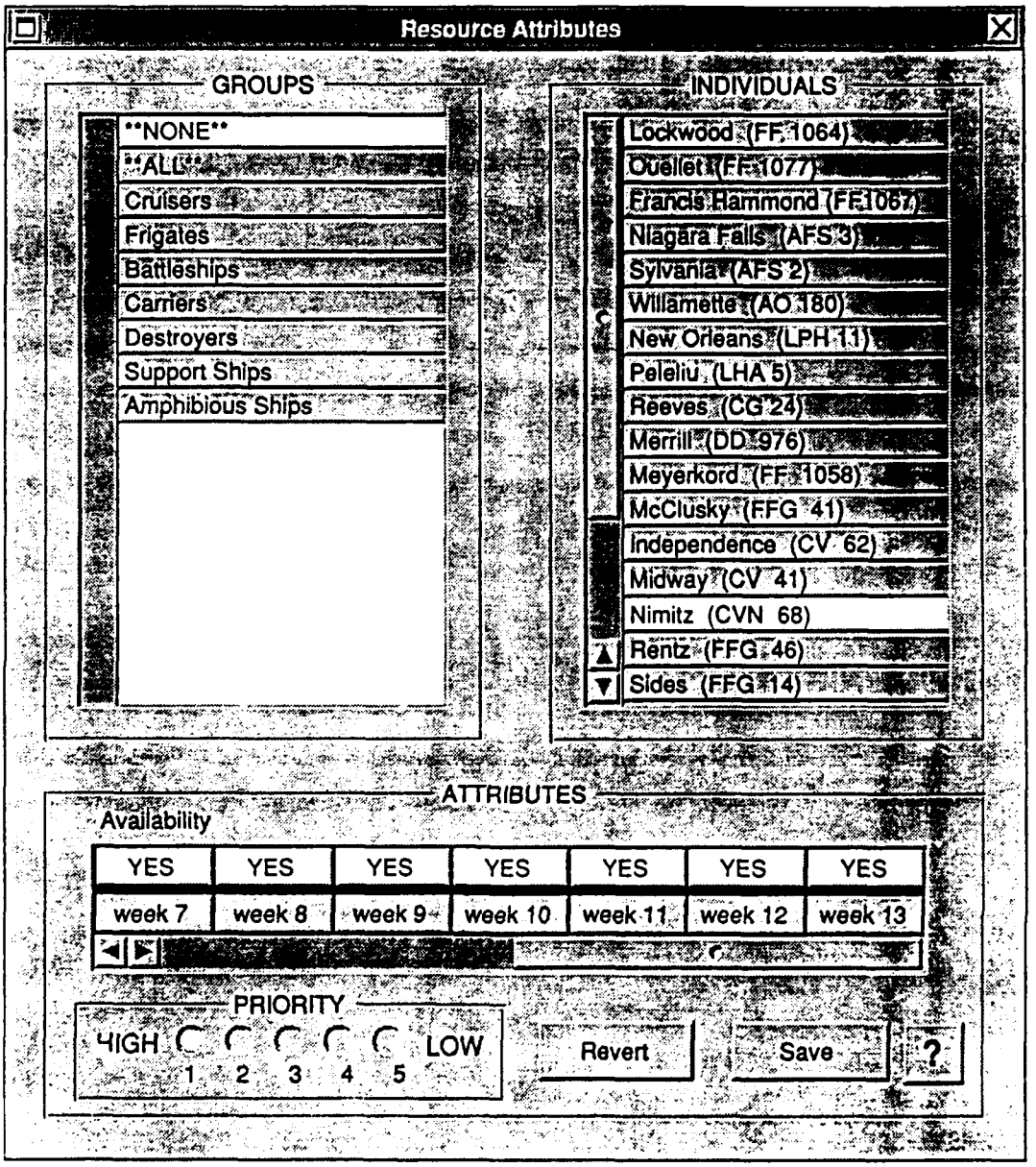


Figure 9. Resource Attributes Window

e. Task Group Assignments

The "Task Group" window, Figure 10, is displayed when "Task" is selected from the "Group" Sub-menu. It is designed to permit the user to perform the same grouping functions for tasks that were performed with resources. All the mechanisms function in the same way as on that window.

f. Task Attribute Assignment

The "Task Attributes" window, Figure 11, is displayed when "Task" is selected from the "Attributes" Sub-menu. It enables the user to assign the simple attribute *Supply* to each task. This can be done individually, or by group in the same manner as with resources. For each week of the schedule the user types in an integer representing the maximum number of highlighted tasks which can be scheduled.

g. Need Identification

The "Need Identification" window, Figure 12, is displayed when "Need" is selected from the Main Menu. It allows the user to identify which of the tasks are actually required by each of the resources. Need can be identified either by individual resource or by group. The "left to right" paradigm is employed in this window with the user first selecting the resource (or resource group) and then identifying the needed tasks. The needed tasks may be indicated individually, or if all the needed tasks are defined by a task group, that group may be selected instead. As before, the selection of a group on either side highlights the members of that group. To save the assignment of needed tasks, the "Save" button is again used. The "Revert" button restores the last saved assignment.

h. Needed Task Attributes

The "Needed Task Attributes" window, Figure 13, is displayed when "Need" is selected from the "Attributes" Sub-menu. It is where the user defines all those attributes that are dependent on both the resource and the task it needs to perform. These attributes are:

- Last Completion
- Duration
- Ideal Periodicity
- Periodicity Slack
- Importance
- Immediate Prerequisites
- Compatible Tasks

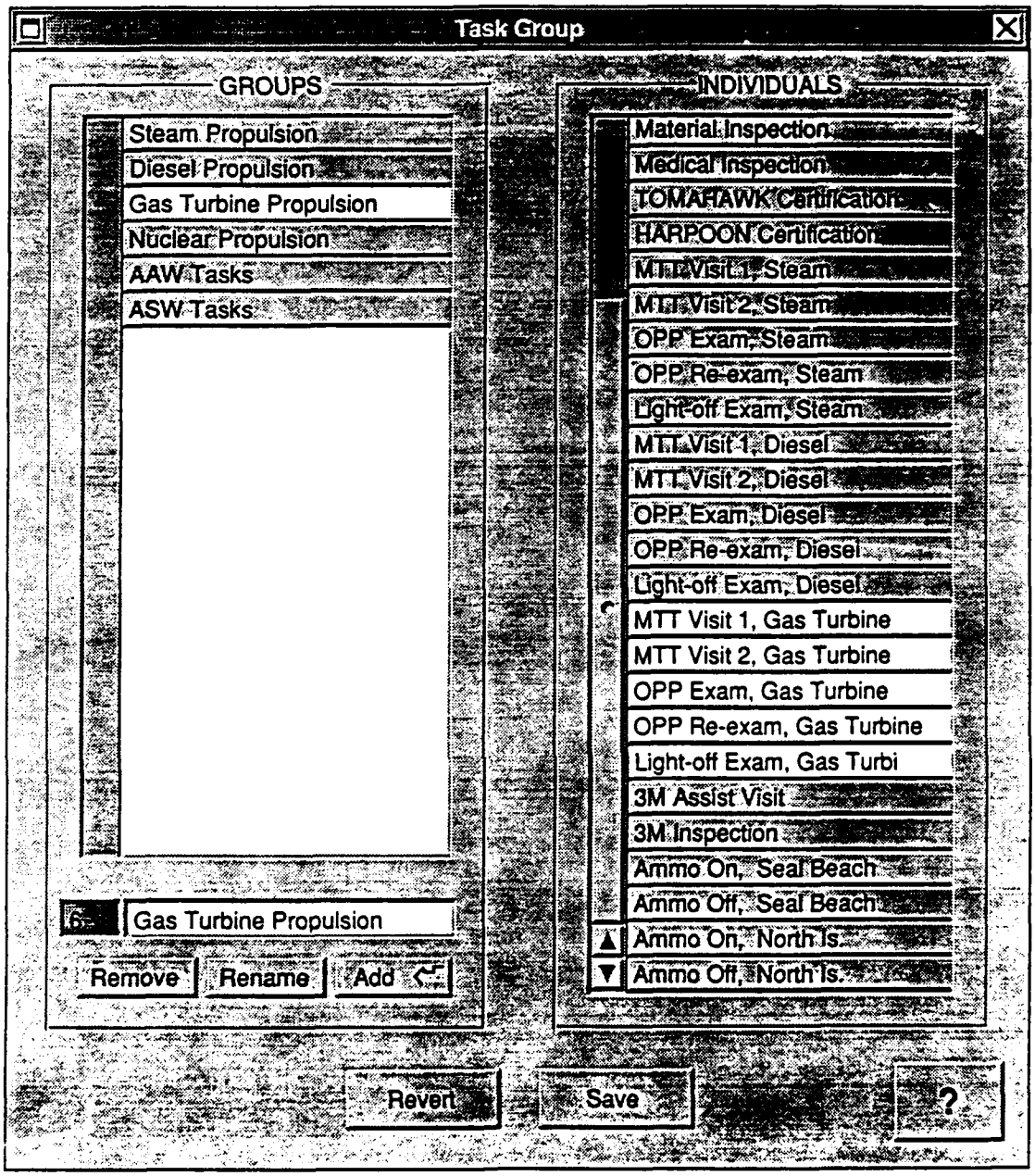


Figure 10. Task Group Window

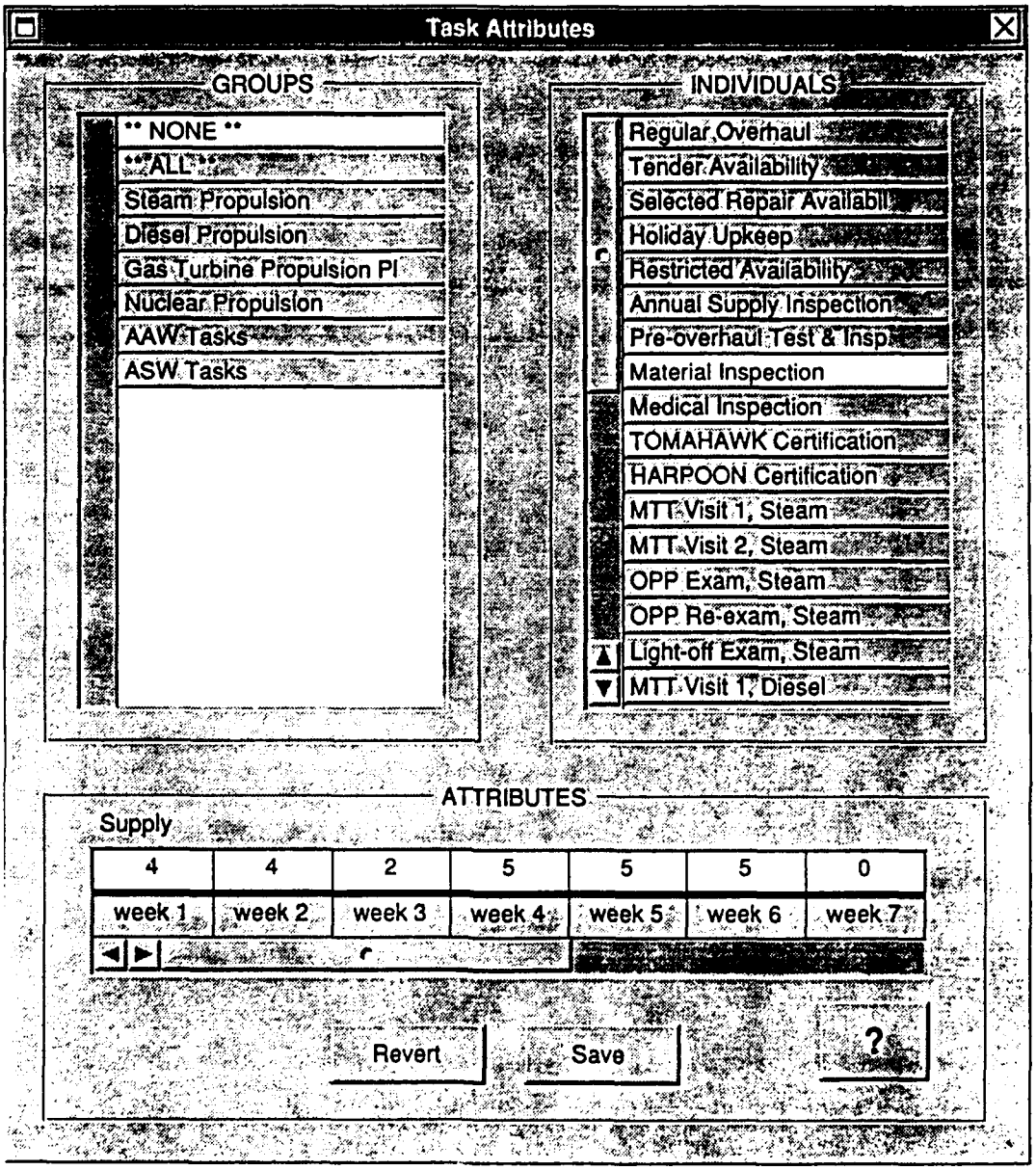


Figure 11. Task Attributes Window

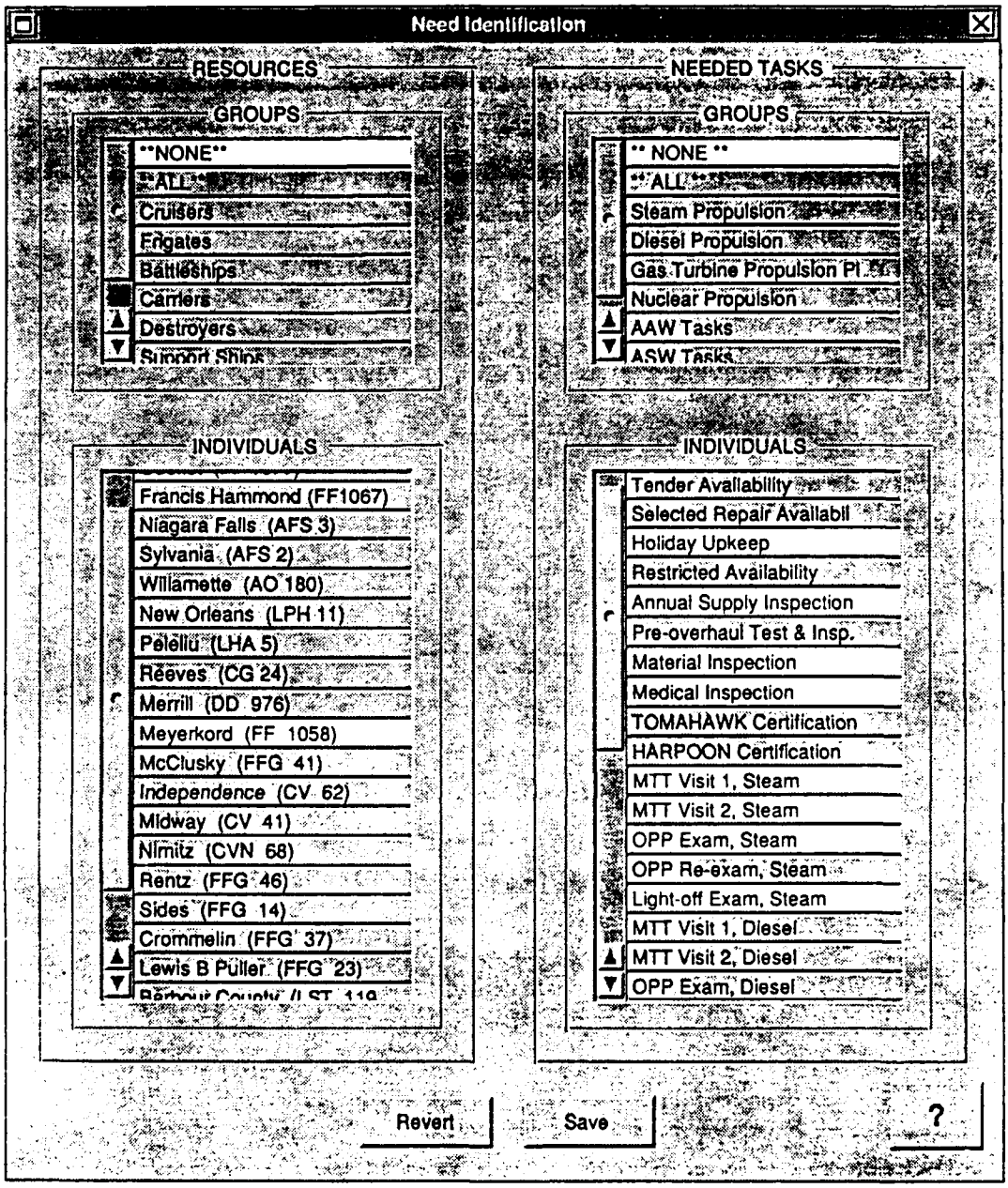


Figure 12. Need Identification Window

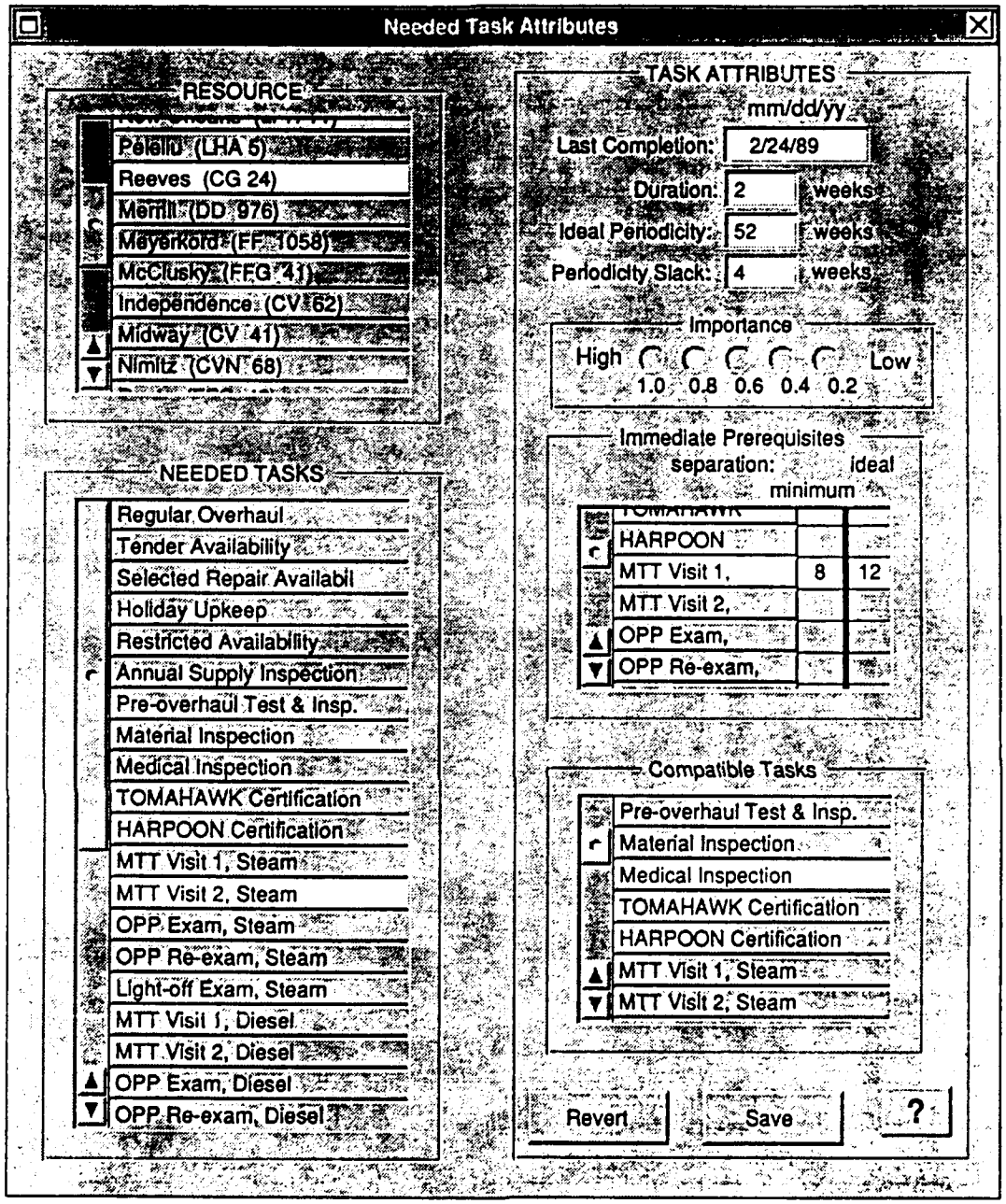


Figure 13. Needed Task Attributes Window

The needed tasks list is empty until the user selects a resource, then the list of needed tasks (previously defined) for that resource are displayed. The user then selects one or more of the needed tasks and assigns values for the various attributes. The user may select one or more prerequisite tasks from the list. Once a prerequisite task is selected, two empty fields appear to the right, allowing the user to define an ideal interval between the task and each prerequisite, and the allowable slack in that ideal, if any. The defaults are zero for both values. The user may also identify other tasks which are compatible (ie. may be performed concurrently). The "Save" and "Revert" buttons have the same functionality as in previous windows.

i. Direct Assignments

The "Lock-ins" window, Figure 14, is displayed when "Lock-ins" is selected from the Main Menu. It allows the user to circumvent the optimization routine and to assign directly any needed tasks to any period in the schedule. The user selects the resource and the needed tasks and their durations are displayed. The user then can select any one of these tasks, click a desired time period and the task will be displayed in that time period and any following time periods if the duration is greater than one. The "Save" and "Revert" buttons store or restore the direct assignments for a give resource.

j. Goal Identification

The user is given the ability to view each of the cost factors and assign weighting factors in the "Goals" window, Figure 15, which is displayed when "Goals" is selected from the Main Menu. When the user selects one of the four goals which the application models, a description is displayed which explains how the cost is computed. The sliders which control the associated weight factors are displayed all the time. This allows the more experienced user, who no longer needs to review the explanation, to make any weight adjustment deemed necessary. The user adjusts the weight factors by moving the associated slider up (increased weight) or down (decreased weight). The range of each of these sliders is from one to two. Under "Settings", the "Save" button is used to save a particular setting of the sliders and the "Retrieve" button allows the user to retrieve a previously stored setting. The "Solve" button is used to initiate the solution process using the settings that are shown. Pressing this button will call up a panel which allows the user to specify output parameters for the optimal schedule which is generated.

k. Schedule Presentation

Although not currently implemented, selecting "Schedule" from the Main Menu will allow the user to view the optimal schedule generated by the solver. Ideally,

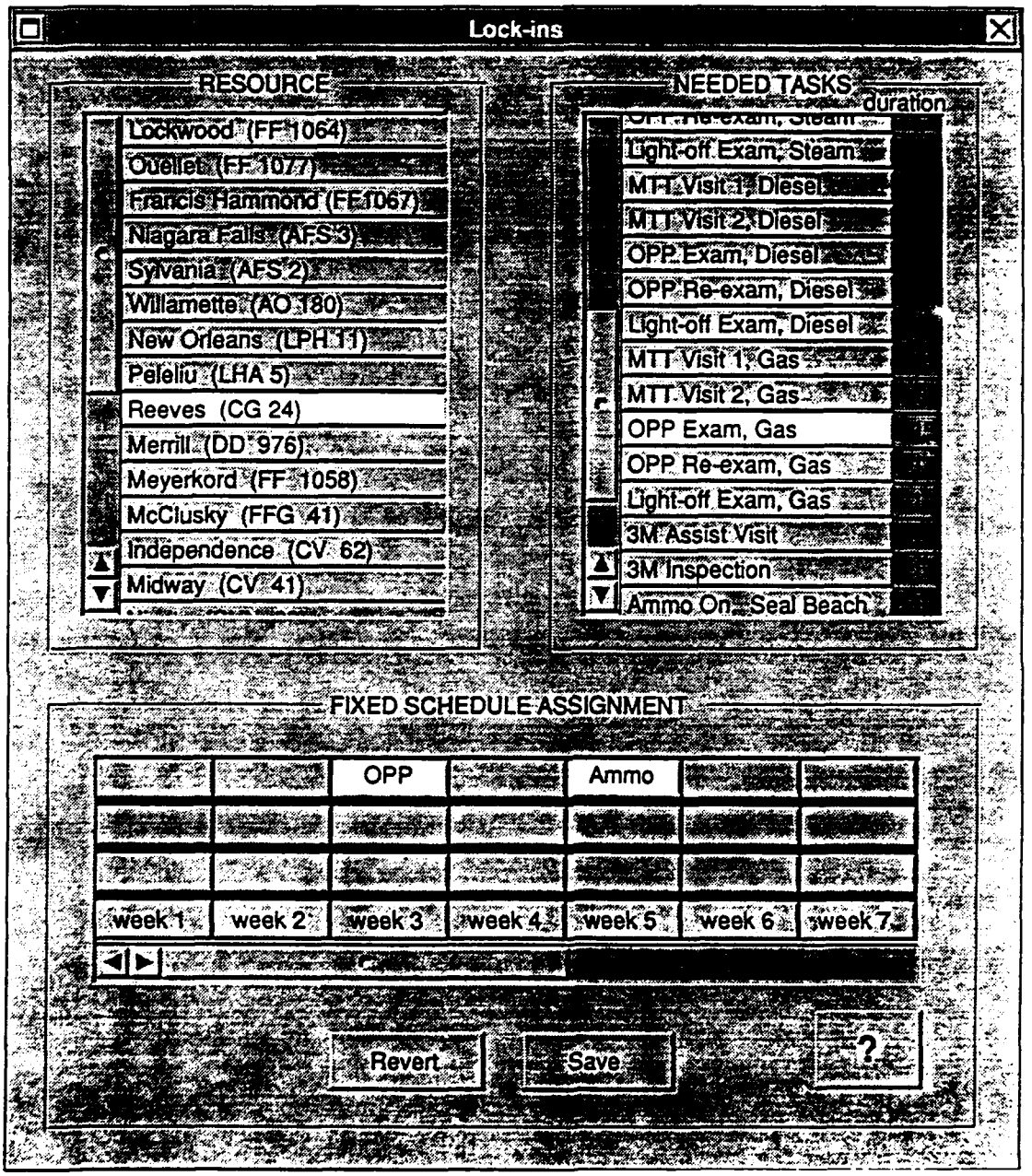


Figure 14. Lock-ins Window

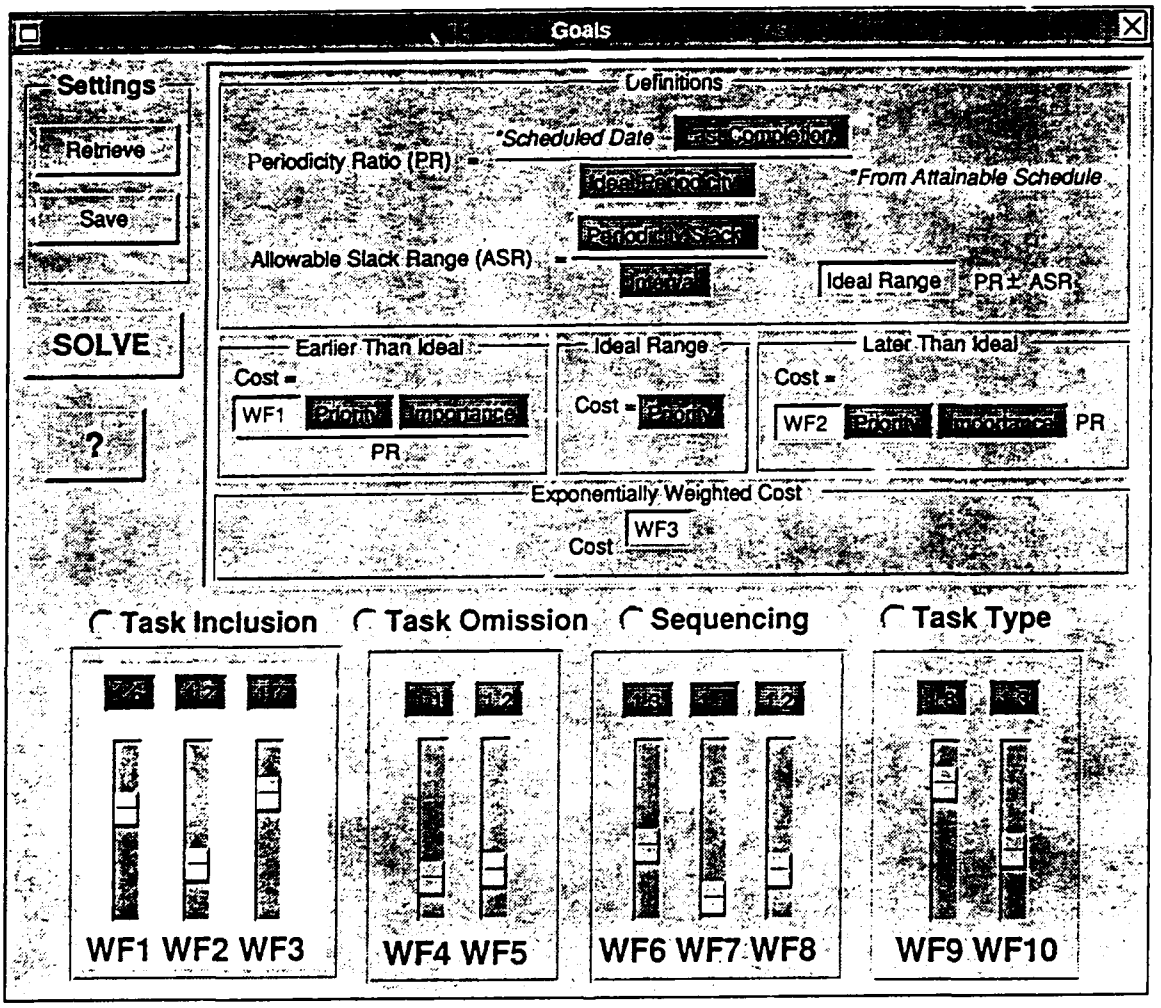


Figure 15. Goals Window

the user will be given a choice of views which allow the schedule information to be analyzed from different perspectives.

D. SUMMARY

The interface components of ***TaskMaster*** that have been designed to this point are those that allow the user to input the data and parameters to the model. The data and parameters which are solicited are based on the generic scheduling model that was presented in Chapter II. Connection with the actual model remains to be accomplished. The ***TaskMaster*** prototype was designed employing the interface principles of Chapter III and the more specific concepts that were formulated during the prototyping process. The ***Taskmaster*** prototype provides an alternative to the fixed formatted files and the fixed model parameters traditionally associated with scheduling models. It also demonstrates how an interface can be constructed generically to accommodate different problem domains. The ***TaskMaster*** prototype is designed to be intuitive enough to be understood by the scheduler with limited OR knowledge and therefore designed to entice schedulers to use the sophisticated OR scheduling tools which many have previously avoided.

V. OBSERVATIONS AND RECOMMENDATIONS

A. OBSERVATIONS

1. Prototyping is an Effective Methodology

In recent years prototyping has gained popularity as an application development methodology. During the development of the ***TaskMaster*** interface it proved to be an invaluable methodology. Interface design is such that it is almost always more effective to view a prototype display than to try to describe it. Prototyping also quickly exposed potential problems and solutions which probably would not have been identified until much later with a more traditional development approach. The benefit of prototyping in this thesis was the ability to not only suggest an approach, but also demonstrate it.

2. Prototype Design is not Easy

There are many different ways of doing essentially the same thing. The difficulty is in identifying the one that will be most effective. This is the essence of much design work. It is often hard to know where to begin, and, once begun, there are the inevitable set backs that occur when a better, more efficient approach is identified. The process of prototyping often involves two steps forward, and one step back. It truly is a learning process. The advantage of prototyping is that this experience is gained early on rather than after the design has been fully (and incorrectly) developed on paper.

The graphical user interface, while greatly enhancing the utility and clarity of the system to the user, exponentially increases the decisions that can be made by the user (ie. size of font, gray scale, style of control mechanism). This has been largely offset by the development of standard graphical objects within the Interface Builder on the NeXT, however, a significant number of choices remain.

3. Reusable Objects Greatly Improve Productivity

Once some useful graphical objects have been designed, the ability to reuse them throughout the design (including slight variations) is a great time-saver. The impact of this approach was observed in the number of lines of code generated. Initial development effort resulted in the creation of a large amount of new code, but as the development progressed beyond that initial stage the increases were much smaller due to reuse of previously developed objects. In addition to the advantage of speeding development, it also results in an interface with a more consistent "look and feel".

B. ADDITIONAL RESEARCH AND DEVELOPMENT

There are at least three areas of additional research which would logically follow from this initial development. These three are discussed briefly below.

1. Extend TaskMaster to Schedule Output

TaskMaster was developed primarily as proof of a concept, however, as an evolutionary prototype it could eventually be developed into a fully functioning DSS. Due to time limitations only model parameterization and data entry were addressed in this first iteration. The full integration of the interface with the solver, and the design of output representations remain. The principles of interface design explored in this thesis could be used to extend the prototype to include these representations. The depiction of large solution sets and providing information on the sensitivity of the solution to the various constraint and objective coefficients are two areas of special importance.

2. Develop the Database Subsystem

This thesis concentrated primarily on the dialog subsystem, or interface, because that has the greatest influence on making the system directly usable by decision makers. The third component of the DSS, the database subsystem needs to be more fully developed to make this a more functional DSS. A relational approach has been suggested and some basic concepts have been mentioned in this thesis which could be extended to the design of a database management system to compliment the model and the dialog.

3. Combine with the CPM Model

It was mentioned in the scheduling chapter that ideally a scheduling DSS would encompass several mathematical models. During the course of designing *TaskMaster*, the basic nature of the scheduling problem was explored for the purpose of identifying a fundamental model structure. This exploration led to a comparison of the set-partitioning solution technique with the CPM model. The CPM model essentially models one constraint; prerequisite relationships. Further, the CPM model itself does not provide a rigorous solution, except for those activities which lie on the critical path. In this way the output of the CPM model may be viewed as a class of problems. Some models add a resource leveling constraint to the output of the CPM solution to identify an optimal solution. The set partitioning aspect of the generic model presented in this thesis could function as a leveling mechanism to be used in concert with CPM. The class of solutions that result from the CPM technique could be fed into the generic model and an optimal solution generated. This would be an interesting topic for further study.

C. CONCLUSION

This thesis was initiated based upon the conviction that there was an advantage to be achieved by putting models directly into the hands of decision makers. This conviction has increased during the course of the research. It became increasingly apparent that the real success of a scheduling system often lies not in providing an optimized solution, but rather in the ability to provide the user with an understanding of the very nature of the process itself. This observation is similar to that made by Goodman:

> Thus, the modeling process provides additional insight into the scheduling problem and results in a standardized method for evaluating a proposed schedule. The ability to critically evaluate and compare alternative proposals is potentially the greatest management tool to be gained from automating the scheduling process through the use of an optimization model. [Ref. 39: p. 150-153].

History shows that the most successful manager is not the one who most accurately models the current constraints, but the one who most effectively uses his understanding of the system to change or eliminate them. Therefore, any system which unnecessarily confuses the decision maker or obscures the vision of the solution process is counterproductive. This objective dictates a different approach to the design of interfaces to these models. They must be designed so that they are understandable and easy to use while still powerful enough to model meaningful problems. They must guide decision makers through the decision process, educating them if necessary in the nuances of the model. And, they should be flexible enough to handle a variety of cases. This thesis and the *TaskMaster* prototype should effectively demonstrate the viability and attractiveness of this approach.

LIST OF REFERENCES

1. O'Dell, D. D., *The Design and Implementaion of a Visual User Interface for a Structured Model Management System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.

2. Woolsey, R. E. D. and Swanson, H. S., *Operations Research for Immediate Application, A Quick and Dirty Approach*, Harper and Row Publishers, 1975.

3. Hackathorn, R., and Keen, P. G. W., "Organizational Strategies for Personal Computing in Decision Support Systems," *Management Information Systems Quarterly*, v. 5, no. 3, pp. 21-27, September 1981.

4. Geoffrion, A. M., "Can MS/OR Evolve Fast Enough?" *Interfaces*, Vol 13, No.1, pp. 10-25, February 1982.

5. Jones, C. V., "The Three-Dimensional Gantt Chart," *Operations Research*, v. 36, no. 6, pp. 891-903, December 1988.

6. Carlson, E. D., and Sprague, R. H., *Building Effective Decision Support Systems*, Prentice Hall, 1982.

7. Brennan, J. J. and Elam, J., "Enhanced Capabilities for Model-Based Decision Support Systems," in R.H. Sprague and H.J. Watson (ed.), *Decision Support Systems, Putting Theory Into Practice*, pp. 130-137, Prentice Hall, 1986.

8. Loyola, S.J., Reilly, N.B., and Werntz, D.G., "Problem-Independent Scheduling Systems," paper presented at ORSA TIMS, New York, New York, 16 October 1989.

9. Geoffrion, A. M., "An Introduction to Structured Modeling," *Management Science*, v. 33, no. 5, pp. 547-588, May 1987.

10. Wing, V. F., *SURFSKED An Optimization Aid for Surface Combatant Inter-Deployment Scheduling*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1986.

11. Baker, K. R., *Introduction to Sequencing and Scheduling*, John Wiley & Sons Inc., 1974.

12. Dolan, M. K., and Kroenke, A. D., *Database Processing; Fundamentals, Design, Implementation*, Chicago, IL, Science Research Associates, 1988.

13. Dolk, D. R., and Konsyski, B. R., "Model Management in Organizations," *Information and Management*, Vol 9, No.1, August 1985.

14. Simpson, H., "A Human-Factors Style Guide for Program Design," *BYTE*, v. 7, no. 4, pp. 108-132, April 1982.

15. Draper, S. W., and Norman, D. A., *User Centered Systems Design: New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates,Inc., 1986.

16. Jones, C. V., "User Interfaces," Working Paper, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Revised December 1988.

17. Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Company, 1987.

18. Bennett, J. L., "Analysis and Design of the User Interface for Decision Support Systems," in J. L. Bennett (ed.), *Building Decision Support Systems*, pp. 41-64, Addison-Wesley, 1983.

19. Card, S. K., Moran, T. P. and Newell, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, 1983.

20. Carroll, J. M., and Rosson, M. B., "Paradox of the Active User," *Interfacing Thought*, Carroll, J. M. (Ed.), pp. 80-111, MIT Press, 1987.

21. Barlow, V. M., Bentley, L. D., and Whitten, J. L., *Systems Analysis and Design Methods*, 2nd ed., Irwin, 1989.

22. Alter, S. L., "Why is Man-Computer Interaction Important for Decision Support Systems," in *Decision Support Systems a Data-Based, Model-Oriented, User-Developed Discipline*, W. C. House (ed.), Petrocelli Books Inc., 1983.

23. Smith, D. C., and others, "Designing the Star User Interface," *BYTE Magazine*, pp. 242-282, v. 7, no. 4, April 1982.

24. Mills, C. B., and Weldon, L. J., "Reading Text from Computer Screens," *ACM Computing Surveys*, v. 19, No.4, pp. 329-358, December 1988.

25. Tufte, E. R., *The Visual Display of Quantitative Information*, Graphics Press, 1983.

26. Brooks, R., "Using A Behavioral Theory of Comprehension in Software Engineering," *The Third International Conference on Software Engineering Proceedings*, pp. 196-201, 1978.

27. Poole, L., "A Tour of the Mac Desktop," *MacWorld*, vol. 1, no. 1, pp.16-21, 1984.

28. Laurel, B. K., "Interfaces as Mimesis," *User Centered Systems Design: New Perspectives on Human-Computer Interaction*, Draper, S. W. and Norman, D. A. (Eds.), Lawrence Erlbaum Associates,Inc., 1986.

29. Baran, N., and Hayes, F., "A Guide to GUIs," *BYTE*, v. 14, no. 7, pp. 250-257, July 1989.

30. Beard, M., and others, "The Xerox Star: A Retrospective," *IEEE Computer*, pp. 11-29, September 1989.

31. Meng, B., "Object-Oriented Programming," *MacWorld*, pp. 174-180, January 1990.

32. Hartson, H. R., "User-Interface Management Control and Communication," *IEEE Software*, v. 6, no. 1, pp. 62-70, January 1989.

33. Calder, P. R., Linton, J. M. and Vlissides, J. M., "Composing User Interfaces with Interviews" *Computer*, v. 22, no. 2, pp. 8-22, February, 1989.

34. Bannon, L. J., Draper, S. W., and Norman, D. A., "Glossary," *User Centered Systems Design: New Perspectives on Human-Computer Interaction*, Draper, S. W., and Norman, D. A. (Eds.), Lawrence Erlbaum Associates,Inc., 1986.

35. Turban, E., *Decision Support and Expert Systems, Managerial Perspectives*, Macmillan Publishing Co., 1988.

36. Hartson, H. R., and Hix, D., "Human-Computer Interface Development," *ACM Computing Surveys*, v. 21, no. 1, pp. 5-92, March 1989.

37. Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1985.

38. Larkin, D., and others, *NeXT Preliminary 1.0 System Reference Manual: Concepts*, NeXT Inc., 900 Chesapeake DR, Redwood City, CA 94063, 1989.

39. Goodman, C. E., *Annual Scheduling of Atlantic Fleet Combatants*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Lieutenant S. R. Banham US Navy Public Works Center FPO San Francisco 96630	2
4.	Professor Gordon H. Bradley, Code ORBZ Naval Postgraduate School Monterey, CA 93943	10
5.	Professor Daniel R. Dolk, Code ASDK Naval Postgraduate School Monterey, CA 93943	3
6.	Computer Technologies Curricular Office, Code 37 Naval Postgraduate School Monterey, CA 93943	1
7.	Commanding Officer Naval Aviation Maintenance Office (NAMO) Code 611 Naval Air Station Patuxent River, MD 20670-5446	1